
21世纪高职高专规划教材·计算机系列
计算机导论
与程序设计
吴学毅 李向前 编著
清华大学出版社
http://www.tup.tsinghua.edu.cn
北京交通大学出版社
http://press.bjtu.edu.cn

责任编辑：谭文芳
特邀编辑：刘汉斌
责任美编：一克米工作室

21世纪高职高专规划教材·计算机系列

计算机应用基础
计算机组成原理
微型计算机原理与接口技术
数据结构概论
数据库系统原理与应用教程
计算机导论与程序设计
计算机组装维护与维修
网络安全与维护
网络操作系统
网页设计技术
动态网页设计
软件工程
SQL Server 实训教程
C语言程序设计教程
C++语言程序设计
Visual Basic .NET程序设计教程
Visual Basic程序设计及实训教程
JAVA程序设计
Flash MX动画设计
Power Builder程序设计
Delphi7.0程序设计教程
ASP .NET使用程序设计
……

ISBN 7-81082-502-X
9 787810 825023 >

ISBN 7-81082-502-X/TP·184
定价：25.00元

21世纪高职高专规划教材·计算机系列

计算机导论与程序设计

吴学毅　李向前　编著

清华大学出版社
北京交通大学出版社
·北京·

内 容 简 介

本书是作者在长期教学实践基础上编写而成的。全书内容分为两篇。第1篇为计算机导论篇，包括计算机基本概念、信息表示、硬件系统组成、软件系统组成、计算机网络构成、Windows操作系统使用、Word和Excel的使用等内容。第2篇为C语言程序设计篇，详细地介绍C语言的基础知识、语法规则和利用C语言进行程序设计的方法等内容。

本书配有大量的例题和习题，连贯性强，并且概念清晰、文字流畅、通俗易懂，具有集计算机导论和C语言程序设计于一书的特色。特别适合于作高职高专计算机专业计算机导论与程序设计课程的教材和参考书，也可以供其他专业的学生阅读和参考。

版权所有，翻印必究。举报电话：010-62782989　13501256678　13801310933
本书封面贴有清华大学出版社防伪标签，无标签者不得销售。
本书防伪标签采用特殊防伪技术，用户可通过在图案表面涂抹清水，图案消失，水干后图案复现；或将表面膜揭下，放在白纸上用彩笔涂抹，图案在白纸上再现的方法识别真伪。

图书在版编目(CIP)数据

计算机导论与程序设计/吴学毅,李向前编著．—北京：清华大学出版社;北京交通大学出版社，2005.5
(21世纪高职高专规划教材·计算机系列)
ISBN 7-81082-502-X

Ⅰ.计…　Ⅱ.①吴…　②李…　Ⅲ.①电子计算机-高等学校:技术学校-教材　②C语言-程序设计-高等学校:技术学校-教材　Ⅳ.TP3

中国版本图书馆CIP数据核字（2005）第019066号

责任编辑：谭文芳　　特邀编辑：刘汉斌
出 版 者：清华大学出版社　　邮编：100084　　电话：010-62776969
　　北京交通大学出版社　　邮编：100044　　电话：010-51686414
印 刷 者：北京瑞达方舟印务有限公司
发 行 者：新华书店总店北京发行所
开　　本：185×260　　印张：18.5　　字数：473千字
版　　次：2005年5月第1版　　2005年5月第1次印刷
书　　号：ISBN 7-81082-502-X/TP·184
印　　数：1～5000册　　定价：25.00元

本书如有质量问题，请向北京交通大学出版社质监组反映。对您的意见和批评，我们表示欢迎和感谢。
投诉电话：010-51686043，51686008；传真：010-62225406；E-mail：press@center.bjtu.edu.cn。

21世纪高职高专规划教材·计算机系列
编审委员会成员名单

主任委员 李兰友　边奠英

副主任委员 周学毛　崔世钢　王学彬　丁桂芝　赵　伟
韩瑞功　汪志达

委　　员 （按姓名笔画排序）

马　辉	万志平	万振凯	王永平	王建明
尤晓玮	丰继林	尹绍宏	左文忠	叶　华
叶　伟	叶建波	付晓光	付慧生	冯平安
江　中	佟立本	刘　炜	刘建民	刘　晶
曲建民	孙培民	邢素萍	华铨平	吕新平
陈国震	陈小东	陈月波	李长明	李　可
李志奎	李　琳	李源生	李群明	李静东
邱希春	沈才梁	宋维堂	汪　繁	吴学毅
张文明	张权范	张宝忠	张家超	张　琦
金忠伟	林长春	林文信	罗春红	苗长云
竺士蒙	周智仁	孟德欣	柏万里	宫国顺
柳　炜	钮　静	胡敬佩	姚　策	赵英杰
高福成	贾建军	徐建俊	殷兆麟	唐　健
黄　斌	章春军	曹豫莪	程　琪	韩广峰
韩其睿	韩　劼	裘旭光	童爱红	谢　婷
曾瑶辉	管致锦	熊锡义	潘玫玫	薛永三
操静涛	鞠洪尧			

出 版 说 明

高职高专教育是我国高等教育的重要组成部分，它的根本任务是培养生产、建设、管理和服务第一线需要的德、智、体、美全面发展的高等技术应用型专门人才，所培养的学生在掌握必要的基础理论和专业知识的基础上，应重点掌握从事本专业领域实际工作的基本知识和职业技能，因而与其对应的教材也必须有自己的体系和特色。

为了适应我国高职高专教育发展及其对教学改革和教材建设的需要，在教育部的指导下，我们在全国范围内组织并成立了“21 世纪高职高专教育教材研究与编审委员会”（以下简称“教材研究与编审委员会”）。“教材研究与编审委员会”的成员单位皆为教学改革成效较大、办学特色鲜明、办学实力强的高等专科学校、高等职业学校、成人高等学校及高等院校主办的二级职业技术学院，其中一些学校是国家重点建设的示范性职业技术学院。

为了保证规划教材的出版质量，“教材研究与编审委员会”在全国范围内选聘“21 世纪高职高专规划教材编审委员会”（以下简称“教材编审委员会”）成员和征集教材，并要求“教材编审委员会”成员和规划教材的编著者必须是从事高职高专教学第一线的优秀教师或生产第一线的专家。“教材编审委员会”组织各专业的专家、教授对所征集的教材进行评选，对列选教材进行审定。

目前，“教材研究与编审委员会”计划用 2～3 年的时间出版各类高职高专教材 200 种，范围覆盖计算机应用、电子电气、财会与管理、商务英语等专业的主要课程。此次规划教材全部按教育部制定的“高职高专教育基础课程教学基本要求”编写，其中部分教材是教育部《新世纪高职高专教育人才培养模式和教学内容体系改革与建设项目计划》的研究成果。此次规划教材编写按照突出应用性、实践性和针对性的原则编写并重组系列课程教材结构，力求反映高职高专课程和教学内容体系改革方向；反映当前教学的新内容，突出基础理论知识的应用和实践技能的培养；适应“实践的要求和岗位的需要”，不依照“学科”体系，即贴近岗位群，淡化学科；在兼顾理论和实践内容的同时，避免“全”而“深”的面面俱到，基础理论以应用为目的，以必要、够用为度；尽量体现新知识、新技术、新工艺、新方法，以利于学生综合素质的形成和科学思维方式与创新能力的培养。

此外，为了使规划教材更具广泛性、科学性、先进性和代表性，我们希望全国从事高职高专教育的院校能够积极加入到“教材研究与编审委员会”中来，推荐“教材编审委员会”成员和有特色、有创新的教材。同时，希望将教学实践中的意见与建议及时反馈给我们，以便对已出版的教材不断修订、完善，不断提高教材质量，完善教材体系，为社会奉献更多更新的与高职高专教育配套的高质量教材。

此次所有规划教材由全国重点大学出版社——清华大学出版社与北京交通大学出版社联合出版。适合于各类高等专科学校、高等职业学校、成人高等学校及高等院校主办的二级职业技术学院使用。

21 世纪高职高专教育教材研究与编审委员会

2004 年 3 月

前　言

《计算机导论与程序设计基础》包括两篇。第1篇是计算机导论，旨在使学生对计算机科学的发展史、学科特点及应用情况有一个概括的了解，同时介绍了计算机系统的硬件、软件的构成和基本功能，以及计算机网络的基础知识和应用。第2篇是C语言程序设计。C语言具有丰富的数据类型。它所提供的数据结构和控制结构适合于进行结构化程序设计，具有使用灵活、可移植性好等特点，并正逐步成为程序设计的入门语言。本篇主要讲述C语言的语法规则和基本概念，通过大量的例题分析，可帮助计算机专业及相关专业的大学一年级学生理解和掌握利用C语言进行程序设计的方法和一些基本算法，对计算机学科的理论基础、应用有一个基本的了解，为更好地步入后续课程的学习打下基础。

由于计算机学科发展很快，故在国内出版的计算机导论教材中，有些内容已经过时，有些内容太深太难，不适合高职、高专相关专业的学生使用。高职、高专类的计算机导论教材既应介绍计算机的基础知识，又要介绍高级语言程序设计方法和基本算法，所以在多年教学实践和对计算机导论教材建设迫切性认识的基础上，我们认真编写了这本适合于高职、高专相关专业学生使用的《计算机导论与程序设计基础》。

本书由14章组成。第1章主要介绍计算机的发展史、应用领域及计算机的基本工作原理、数制转换和内部编码；第2章主要介绍计算机硬件系统各部件的工作原理；第3章主要讨论计算机的软件系统，着重对操作系统、数据库管理系统和计算机语言进行了讨论；第4章主要讲述计算机网络的基本原理和网络设备的连接方法；第5章主要讲述目前流行的Windows操作系统、文字编辑软件Word和电子表格软件Excel的使用和操作方法；第6章～第14章主要讨论C语言程序设计的基本概念、程序设计方法和实现及一些基本算法。

由于本书介绍了计算机方面的许多基本概念和基本术语，同时又讲述C语言程序设计的基本方法和一些基本算法，因此不仅可以作为计算机导论与程序设计专业的教材使用，同时又可作为计算机方面的科普读物。

为了便于学生练习，本书每一章最后均给出了比较充足的习题。

根据编者的教学体会，使用本教材授课约需90～100学时。

本书由吴学毅、李向前编著，在编写过程中得到了付凯、白书弟、宋真君、赵红岩、岳经纬、李中跃等老师及谭文芳编辑的帮助，在此表示衷心的感谢。

由于编者水平有限，书中难免有疏漏之处，恳请读者批评指正。

编　者

2005年3月

目　　录

第 1 篇　计算机导论

第 1 章　计算机的基础知识 …… 3
1.1　计算机概述 …… 3
1.1.1　什么是计算机 …… 3
1.1.2　计算机的发展 …… 3
1.1.3　计算机的发展过程 …… 3
1.1.4　计算机发展的趋势 …… 4
1.1.5　计算机的应用 …… 5
1.2　计算机系统的基本组成和工作原理 …… 5
1.2.1　计算机系统的基本组成 …… 5
1.2.2　计算机的基本工作原理 …… 7
1.3　计算机中的数制及编码 …… 7
1.3.1　进位制数及相互转换 …… 7
1.3.2　计算机中数的表示 …… 10
1.3.3　计算机中的编码 …… 13
小结 …… 16
习题 …… 16
第 2 章　计算机硬件基础 …… 17
2.1　中央处理器 …… 17
2.1.1　运算器 …… 17
2.1.2　控制器 …… 18
2.2　存储器 …… 19
2.2.1　存储器概述 …… 19
2.2.2　半导体存储器 …… 21
2.2.3　磁介质存储器 …… 23
2.2.4　光介质存储器 …… 27
2.3　输入输出设备及接口 …… 27
2.3.1　输入设备 …… 28
2.3.2　输出设备 …… 29
2.3.3　输入输出设备接口 …… 30
小结 …… 32

习题 …… 32
第3章 计算机系统的软件 …… 33
3.1 计算机软件概述 …… 33
3.1.1 软件的概念 …… 33
3.1.2 系统软件 …… 33
3.1.3 应用软件 …… 35
3.2 程序设计语言 …… 36
3.2.1 机器语言 …… 36
3.2.2 汇编语言 …… 37
3.2.3 高级语言 …… 38
3.2.4 最常见的高级语言及其主要应用领域 …… 39
3.3 操作系统 …… 40
3.3.1 操作系统的基础知识 …… 40
3.3.2 资源管理中的基本方法 …… 42
3.4 数据库系统 …… 47
3.4.1 数据库系统概述 …… 47
3.4.2 数据库管理系统的组成 …… 48
3.4.3 数据库管理系统的分类 …… 50
3.4.4 常用关系数据库管理系统 …… 52
小结 …… 53
习题 …… 53
第4章 计算机网络系统 …… 55
4.1 计算机网络概述 …… 55
4.1.1 计算机网络的发展历程 …… 55
4.1.2 计算机网络的概念 …… 56
4.1.3 计算机网络协议 …… 56
4.2 计算机网络的分类 …… 57
4.2.1 按网络的范围大小分类 …… 57
4.2.2 按网络的拓扑结构分类 …… 59
4.2.3 按网络的交换方式分类 …… 60
4.3 局域网 …… 61
4.3.1 局域网的组成 …… 61
4.3.2 局域网的连接设备 …… 62
4.3.3 局域网的结构 …… 63
4.4 计算机局域网实例 …… 63
小结 …… 65
习题 …… 65
第5章 计算机的基本操作 …… 66
5.1 Windows 98 操作 …… 66

5.1.1 Windows 98 概述 …… 66
5.1.2 桌面简介 …… 67
5.1.3 Windows 98 文件管理 …… 72
5.2 中文版 Word 操作 …… 78
5.2.1 中文版 Word 简介 …… 78
5.2.2 文档的基本编辑 …… 81
5.2.3 文档的格式化 …… 88
5.2.4 表格的基本操作 …… 93
5.2.5 简单的图文操作 …… 96
5.2.6 预览与打印 …… 98
5.2.7 帮助系统的使用 …… 100
5.3 Excel 操作 …… 100
5.3.1 Excel 概述 …… 100
5.3.2 编辑工作表数据 …… 102
5.3.3 格式化工作表 …… 108
5.3.4 数据管理和分析 …… 111
小结 …… 113
习题 …… 113

第 2 篇 C 语言程序设计

第 6 章 C 语言概述 …… 117
6.1 C 语言简介 …… 117
6.1.1 C 语言的发展历程 …… 117
6.1.2 C 语言的特点 …… 117
6.2 C 源程序的基本结构 …… 118
小结 …… 119
习题 …… 120
第 7 章 算法及其描述方法 …… 121
7.1 算法概述 …… 121
7.2 常用算法的举例 …… 122
7.3 算法的描述方法 …… 124
7.3.1 用自然语言描述算法 …… 124
7.3.2 用传统流程图描述算法 …… 124
7.3.3 用 N-S 流程图描述算法 …… 125
7.3.4 伪代码描述算法 …… 128
7.3.5 用计算机语言描述算法 …… 129
小结 …… 130
习题 …… 130

第 8 章　数据类型、运算符与表达式 …… 131
8.1　C 语言的基本语法单位 …… 131
8.2　常量与变量 …… 132
8.2.1　常量 …… 132
8.2.2　变量 …… 132
8.3　基本数据类型 …… 133
8.3.1　整型数据 …… 133
8.3.2　实型数据 …… 135
8.3.3　字符型数据 …… 136
8.4　变量的初始化 …… 139
8.5　运算符和表达式 …… 139
8.5.1　算术运算符和算术表达式 …… 140
8.5.2　赋值运算符和赋值表达式 …… 141
8.5.3　关系运算符和关系表达式 …… 143
8.5.4　逻辑运算符和逻辑表达式 …… 143
8.5.5　自增、自减运算符 …… 144
8.6　位操作运算符及表达式 …… 146
8.7　条件运算符和条件表达式 …… 148
8.8　逗号运算符和逗号表达式 …… 149
8.9　运算符的结合性和优先级 …… 149
小结 …… 150
习题 …… 150
第 9 章　结构化程序设计 …… 152
9.1　C 语句概述 …… 152
9.2　顺序结构程序设计 …… 154
9.2.1　实现顺序结构程序设计的基本语句 …… 154
9.2.2　顺序结构程序设计举例 …… 161
9.3　选择结构程序设计 …… 162
9.3.1　if 语句 …… 162
9.3.2　switch 语句 …… 167
9.3.3　选择结构程序设计举例 …… 168
9.4　循环结构程序设计 …… 170
9.4.1　while 语句 …… 171
9.4.2　for 语句 …… 172
9.4.3　do...while 语句 …… 174
9.4.4　几种循环的比较 …… 175
9.4.5　break 语句和 continue 语句 …… 176
9.4.6　循环嵌套 …… 177
9.4.7　循环结构程序设计举例 …… 178

小结 …… 180
习题 …… 181
第 10 章 数组 …… 184
10.1 一维数组 …… 184
10.1.1 一维数组的定义 …… 184
10.1.2 一维数组的引用 …… 185
10.1.3 一维数组的初始化 …… 186
10.1.4 一维数组应用举例 …… 186
10.2 二维数组 …… 188
10.2.1 二维数组的定义 …… 188
10.2.2 二维数组元素的引用 …… 189
10.2.3 二维数组的初始化 …… 189
10.2.4 二维数组应用举例 …… 190
10.3 字符数组与字符串 …… 192
10.3.1 字符数组的定义 …… 192
10.3.2 字符数组的初始化 …… 192
10.3.3 字符串和字符数组 …… 193
10.3.4 字符串处理函数 …… 194
小结 …… 195
习题 …… 195
第 11 章 函数 …… 197
11.1 函数的定义 …… 197
11.1.1 函数的概述 …… 197
11.1.2 函数的定义 …… 198
11.1.3 函数参数和返回值 …… 199
11.2 函数的调用 …… 200
11.2.1 函数的调用 …… 200
11.2.2 函数的声明 …… 201
11.3 函数的嵌套调用和递归调用 …… 202
11.3.1 函数的嵌套调用 …… 202
11.3.2 函数的递归调用 …… 203
11.4 函数调用的数据传递方式 …… 205
11.5 局部变量与全局变量 …… 206
11.5.1 变量的作用域 …… 206
11.5.2 变量存储类别 …… 208
11.5.3 局部变量的存储类别 …… 208
11.5.4 外部变量的存储类型 …… 209
11.6 内部函数和外部函数 …… 211
11.6.1 内部函数 …… 211

11.6.2 外部函数 …… 211
11.7 编译预处理 …… 212
11.7.1 宏替换 …… 212
11.7.2 文件包含 …… 213
11.7.3 条件编译 …… 213
小结 …… 214
习题 …… 214
第 12 章 指针 …… 216
12.1 概述 …… 216
12.1.1 存储单元与地址基本概念 …… 216
12.1.2 指针与指针变量的概念 …… 217
12.2 指针变量的使用 …… 217
12.2.1 指针变量的定义与初始化 …… 217
12.2.2 指针的两个基本运算 …… 218
12.3 指针与数组 …… 220
12.3.1 指向数组元素的指针变量与一维数组 …… 220
12.3.2 指针与二维数组 …… 222
12.3.3 指针与字符串 …… 225
12.4 指针与函数 …… 226
12.4.1 指向函数的指针变量 …… 227
12.4.2 返回指针值的函数 …… 229
12.5 指针数组与指向指针的指针变量 …… 229
12.5.1 指针数组 …… 230
12.5.2 指向指针的指针变量 …… 232
12.5.3 用指针数组作为 main()函数的参数 …… 233
小结 …… 234
习题 …… 235
第 13 章 结构体与共用体 …… 238
13.1 结构体 …… 238
13.1.1 定义结构体类型的一般形式 …… 238
13.1.2 结构体变量的定义 …… 239
13.1.3 结构体变量的使用 …… 241
13.2 结构体数组 …… 243
13.2.1 结构体数组的定义 …… 243
13.2.2 结构体数组的引用 …… 244
13.3 指向结构体类型数据的指针 …… 245
13.3.1 指向结构体变量的指针定义 …… 245
13.3.2 用指针引用结构体成员 …… 246
13.4 共用体 …… 248

13.4.1 共用体类型的定义 …… 248
13.4.2 共用体变量的定义 …… 249
13.4.3 共用体变量的引用 …… 249
13.5 枚举类型 …… 251
13.5.1 枚举类型的定义 …… 251
13.5.2 枚举类型变量的定义和使用 …… 251
13.6 用户定义类型 …… 252
13.6.1 用户自定义类型的含义和格式 …… 252
13.6.2 用 typedef 声明基本类型 …… 253
13.6.3 用 typedef 声明构造类型 …… 253
小结 …… 254
习题 …… 255
第 14 章 文件 …… 256
14.1 C 文件概述 …… 256
14.2 文件类型指针 …… 257
14.3 文件的打开与关闭 …… 258
14.3.1 文件的打开(fopen 函数) …… 258
14.3.2 文件的关闭(fclose 函数) …… 260
14.4 文件的读写 …… 260
14.4.1 fputc 函数和 fgetc 函数 …… 260
14.4.2 fread()函数和 fwrite()函数 …… 262
14.4.3 fprintf 函数和 fscanf 函数 …… 264
14.4.4 其他读写函数 …… 265
14.5 文件定位 …… 266
14.6 出错的检测 …… 267
小结 …… 267
习题 …… 268
附录 A 运算符和结合性 …… 269
附录 B C 库函数 …… 271
附录 C Turbo C 常见的出错信息 …… 277
参考文献 …… 280

第 1 篇

计算机导论

第 1 章　计算机的基础知识

本章要点

- ☑ 计算机的发展历程和发展趋势
- ☑ 计算机系统的基本组成
- ☑ 计算机的工作原理
- ☑ 计算机中使用的数制相互转换
- ☑ 计算机中数的表示
- ☑ 计算机中的编码

随着计算机技术的飞速发展，计算机的应用越来越广泛，现已深入到科学研究、工业生产及文化教育等各个领域，特别是微型计算机，其发展和应用对人类社会产生的影响就更加深刻。因此，掌握计算机基础知识及其应用技术已成为当代大学生知识结构的重要组成部分。

1.1　计算机概述

1.1.1　什么是计算机

计算机是一种能够高速计算，具有内部存储能力，由程序控制其操作过程的电子设备。由于计算机能够模仿人脑的功能，如记忆、分析、判断、分类及推理等，可参与一些复杂的科学计算、信息处理和辅助设计，所以人们形象地称计算机为“电脑”。

1.1.2　计算机的发展

世界上第一台计算机于 1946 年诞生在美国，并命名为 ENIAC（Electronic Numerical Integrator And Calculator），由美国宾夕法尼亚大学研制成功。它是一个庞然大物，由 18800 多个电子管、1500 多个继电器及 30 个操作控制台组成，占地面积为 170 m^2，重量为 30 多吨，每小时耗电为 150 kW，运算能力：每秒可进行 5000 次加法运算，56 次乘法运算，比人的运算速度快 20 万倍。该台计算机在美国陆军弹道研究所运行了约 10 年。ENIAC 的诞生为今天计算机科学的发展奠定了基础。

1.1.3　计算机的发展过程

计算机经过 50 多年的发展，不仅在技术上，更是在应用上都是令人鼓舞的。其发展经过了“四代”。这四代的发展体现在以下五个方面。

一是计算机硬件方面，主要是元器件的发展，从电子管元器件发展到晶体管元器件，再到小规模集成电路、中规模集成电路、大规模集成电路及超大规模集成电路。硬件的发展还表现在从简单的外部设备（仅提供简单的输入输出设备）到多样化的外部设备的发展，如键盘、鼠标、数字化仪、扫描仪、音频输入器、手写输入设备、显示器、打印机、绘图仪及音频输出等。

二是运算速度方面，从每秒可进行几十次的运算发展到每秒可进行几万次、几十万次，乃至数千万亿次的运算。

三是系统软件方面，从裸机（不提供任何软件）发展到提供管理程序、操作系统、语言系统、数据库管理系统、网络软件系统及各种软件工具等。

四是计算机应用方面，从单一的科学计算应用发展到数据处理、图像处理及音频处理等应用，使计算机的应用领域从单纯的科学研究领域发展到社会上的几乎所有领域。

五是计算机技术的发展速度方面，计算机技术的发展周期越来越短，硬件的更新周期从五年缩短到两年、一年、八个月，直到现在的两三个月；软件的发展周期从十年缩短到五年、一年。

计算机硬件的发展过程见表 1-1。

表 1-1 计算机硬件的发展过程

阶 段	年 代	主要元器件	特 点
第一代	1946—1957	电子管	速度低，功耗大，价格昂贵，可靠性差，用机器语言编程，应用难度大，仅应用于数值计算
第二代	1958—1964	晶体管	体积缩小，功耗降低，速度增快，价格比较便宜，可以使用高级语言编程，形成软件控制，应用于数据处理和实时控制
第三代	1960—1972	小规模集成电路（SSI） 中规模集成电路（MSI）	体积进一步缩小，速度进一步提高，价格进一步降低，可以使用多种高级语言编程，软件逐步完善，操作系统形成并复杂程度高、功能强大，应用领域迅速扩大
第四代	1970 之后	大规模集成电路（LSI） 超大规模集成电路（VLSI）	微型计算机出现，性能大大提高，价格大大下降，软件更加丰富，应用领域更加扩大，计算机网络普及，小巨型机开始产生

1.1.4 计算机发展的趋势

计算机发展的现实展示了总的发展趋势，即巨型化、微型化、网络化及智能化。

1. 巨型化

巨型化是指发展高速度、大存储容量及强功能的超大型计算机。这主要是为了满足军事、天文、气象、原子、航天、核反应、遗传工程及生物工程等学科研究的需要，同时也可满足计算机人工智能与知识工程研究的需要。

2. 微型化

微型化是指计算机的体积越来越小，功能越来越强，价格越来越低，软件越来越丰富，系统集成度越来越高，操作使用越来越方便。

3. 网络化

用通信线路及通信设备将分散运行的单台计算机连接在一起而形成的一个复杂系统就是计算机网络。这种方式扩大了计算机系统的规模，实现了计算机资源（硬件资源和软件资源）的共享，提高了计算机系统的协同工作能力，为电子数据交换提供了条件。

4. 智能化

智能化是指计算机具有模仿人类较高层次智能活动的能力：模拟人类的感觉、行为、思维过程；具有“视觉”、“听觉”、“说话”、“行为”、“思维”、“推理”、“学习”、“定理证明”、“语言翻译”等的能力。

1.1.5　计算机的应用

在信息化的社会中，计算机的应用十分广泛，主要应用于以下几个方面。

1. 科学计算

在航空、航天、核物理等许多科学领域，都需要进行复杂的运算，而计算机的运算速度和精度是其他任何计算工具所无法比拟的。

2. 数据处理

计算机可以在短时间内对大量的各种各样的的数据进行处理，可满足信息时代的要求。

3. 自动控制

自动控制是在工业生产过程中，对控制对象实现自动控制和自动调节的控制方式，又被称为过程控制。

4. 计算机辅助系统

计算机辅助系统包括计算机辅助设计、计算机辅助制造及计算机辅助教学。

计算机辅助设计是利用计算机来帮助设计人员进行工程或产品设计，以实现最佳设计效果的一种技术。

计算机辅助制造是指利用计算机进行生产设备的管理、控制和操作。

计算机辅助教学是指利用计算机来协助教师进行教学。

5. 计算机网络

计算机网络是利用通信线路和通信设备将分布在不同地理位置上的具有独立功能的数台计算机或终端连接起来，在软件的控制下，实现计算机资源共享和通信。

1.2　计算机系统的基本组成和工作原理

1.2.1　计算机系统的基本组成

从系统的组成观点来看，一个计算机系统应包括硬件系统和软件系统。

1. 硬件系统

硬件系统是指计算机系统各种装置设备的总称。它包括控制器、运算器、存储器、输入设备和输出设备，如图 1-1 所示。

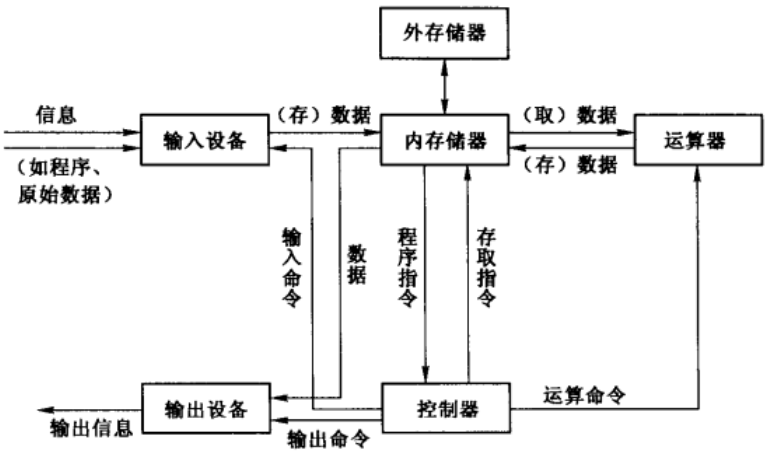


图 1-1 计算机硬件系统结构图

（1）控制器

控制器是整个计算机的中枢神经。其功能是对程序规定的控制信息进行解释，并根据其要求进行控制，调度程序、数据、地址，协调计算机各部分的工作及内存与外设的访问等。

（2）运算器

运算器的功能是对数据进行各种算术运算和逻辑运算，即对数据进行加工处理。

（3）存储器

存储器的功能是存储程序、数据、各种信号及命令等信息，并在需要时提供这些信息。

（4）输入设备

输入设备是计算机的重要组成部分。输入设备与输出设备被合称为外部设备，简称外设。输入设备的作用是将程序、原始数据、文字、字符、控制命令或现场采集的数据等信息输入到计算机。常见的输入设备有键盘、鼠标器、光电输入机、磁带机、磁盘机及光盘机等。

（5）输出设备

输出设备与输入设备一样是计算机的重要组成部分。它把计算机的中间结果或最后结果、机内的各种数据符号及文字或各种控制信号等信息输出来。计算机常用的输出设备有显示终端 CRT、打印机、绘图仪、磁带机及光盘机等。

2．软件系统

与硬件设备相对应的是软件。所谓软件，是用来管理、控制与规定计算机运行步骤的各种程序的总称。软件包括系统软件和应用软件。

使用和管理计算机的各种软件被统称为系统软件。它通常是厂商作为机器产品与硬件同时提供给用户的。计算机配置的基本系统软件包括操作系统、各种高级语言处理程序、编译系统及数据库管理系统四类。

应用软件是由用户利用计算机及其系统软件编制的解决实际应用问题的程序，对使用计算机的人员来说，只有在必要时才需对系统软件进行扩展。目前，应用软件已逐步标准化、模块化和商品化。

1.2.2　计算机的基本工作原理

为了便于理解计算机的基本工作原理，下面将通过一个简单的计算题来说明有关计算机程序的基本概念，首先使用一道简单的计算题来回想一下人脑的工作方式。

题目：8+4÷2=?

首先，用笔将这道题记录在纸上，记在大脑中，再经过脑神经元的思考，结合以前掌握的知识，决定用四则运算规则和九九乘法口诀来处理，即先用大脑计算出 4÷2=2 这一中间结果，并记录于纸上，然后再用大脑计算出 8+2=10 这一最终结果，并记录于纸上。

通过做这一简单的计算题，可发现一规律：首先通过眼、耳等感觉器官将捕捉的信息输送到大脑中并存储起来，然后对这一信息进行加工处理，再由大脑控制人把最终结果，以某种方式表达出来。

计算机正是模仿人脑进行工作的（这也是“电脑”名称的来源），先通过输入设备，如用键盘、鼠标输入指令，然后再由 CPU 来处理这些指令，最后才由输出设备输出计算的结果。其部件，如输入设备、存储器、运算器、控制器及输出设备等分别与人脑的各种功能器官对应，以完成信息的输入、处理及输出。

1.3　计算机中的数制及编码

在计算机内部，数据都是采用二进制数 0 和 1 来表示的，按数据的类型，可分为数值数据和非数值数据两大类。数值数据用于表示数量的大小，常常参加各种数学及逻辑运算。非数值数据指声音、图像及字符类型的数据。

1.3.1　进位制数及相互转换

1. 进位制数

在日常生活中，人们都采用十进制数（decimal number）来表示数值，而在计算机领域中采用的是二进制、八进制（octal）或十六进制数（hexadecimal number）来表示数值。若把它们统称为 R 进制，则该进位制具有下列性质：

- 在 R 进制中，具有 R 个数字符号，它们是 0，1，2，…，(R－1)；
- 在 R 进制中，由低位向高位是按“逢 R 进一”的规则进行计数的；
- R 进制的基数（base）是“R”。R 进制数的第 i 位的权（weight）为“R^i”，并约定整数最低位的位序号 $i=0$（$i=n$，…，2，1，0，－1，－2，…）。

表 1-2 给出了十进制、二进制、八进制及十六进制的特性。由表 1-2 可知，不同进位制具有不同的“基数”；对于某一进位制数，不同的数位具有不同的“权”。基数和位权是进位制数的两个要素。基数表明了某一进位制的基本特征，如对于二进制，有两个数字符号(0，1)，且由低位向高位是“逢二进一”，故其基数为 2。位权表明了一数字符号处于不同数位时所代表值的不同，如对于下列二进制数，各位的“权”为：

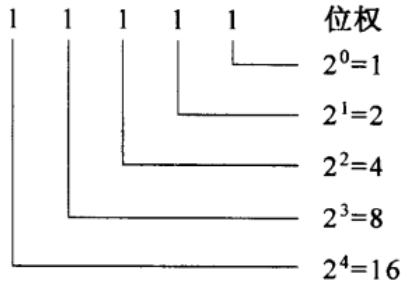


表 1-2 中，用圆括号外的下标值（如 10，2，8，16）表示该括号内的数是哪一个进位制中的数，或在数的最后加上字母 D（十进制）、B（二进制）、Q（八进制）、H（十六进制）来区分其前面的数是属于哪个进位制的。

表 1-2 十进制、二进制、八进制及十六进制的特性

项目 \ 进位制	十 进 制	二 进 制	八 进 制	十六进制
特 点	(1) 具有 10 个数字符号 0，1，2，…，9 (2) 按“逢十进一”的规则计数 (3) 基数为 10，第 i 位的权为 10^i	(1) 具有两个数字符号 0，1 (2) 按“逢二进一”的规则计数 (3) 基数为 2，第 i 位的权为 2^i	(1) 具有 8 个数字符号 0，1，2，…，8 (2) 按“逢八进一”的规则计数 (3) 基数为 8，第 i 位的权为 8^i	(1) 具有 16 个数字符号 0，1，2，…，F (2) 按“逢十六进一”的规则计数 (3) 基数为 16，第 i 位的权为 16^i
举 例	$(199.34)_{10}=1\times10^2+9\times10^1+9\times10^0+3\times10^{-1}+4\times10^{-2}$	$(1011.11)_2=1\times2^3+0\times2^2+1\times2^1+1\times2^0+1\times2^{-1}+1\times2^{-2}$	$(14.76)_8=1\times8^1+4\times8^0+7\times8^{-1}+6\times8^{-2}$	$(19A.6)_{16}=1\times16^2+9\times16^1+10\times16^0+6\times16^{-1}$
表示方法	$(199.34)_{10}=199.34D$	$(1011.11)_2=1011.11B$	$(14.76)_8=14.76Q$	$(19A.6)_{16}=19A.6H$

2. 进位制数的相互转换

同一个数值可以用不同的进位制数表示，这表明不同的进位制只是表示数的不同手段，它们之间一定可以相互转换。下面将通过具体例子说明计算机中常用的几种进位制数之间的转换，即十进制与二进制之间的转换，二进制与八进制之间的转换。

(1) 二进制数转换为十进制数

二进制数转换为十进制数的基本方法是将二进制数的各位按权展开相加。

[例 1-1] 将二进制数 1011.101 转换为十进制数。

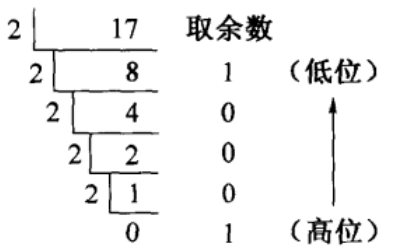
$$\begin{aligned}(1011.101)_2 &=1\times2^3+0\times2^2+1\times2^1+1\times2^0+1\times2^{-1}+0\times2^{-2}+1\times2^{-3}\\ &=8+2+1+0.5+0.125\\ &=(11.625)_{10}\end{aligned}$$

(2) 十进制数转换为二进制数

十进制数转换为二进制数的基本方法是对于整数采用“除 2 取余”，对于小数采用“乘 2 取整”。

[例 1-2] 计算$(17)_{10}=(?)_2$。

采用除 2 取余的计算过程如下：

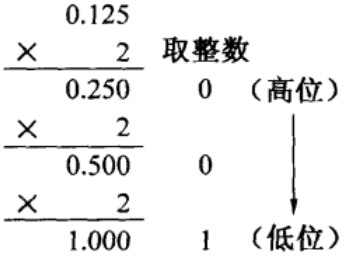


求得 $(17)_{10}=(10001)_2$

由上式可知，用除 2 取余法实现十进制整数的转换规则是：用 2 连续整除要转换的十进制数及各次所得之商，一直除到商为 0 时止，则各次所得的余数即为所求二进制数由低位到高位的值。

[例 1-3]　计算$(0.125)_{10}=(?)_2$。

采用乘 2 取整的计算过程如下：



求得 $(0.125)_{10}=(0.001)_2$

可见，用乘 2 取整法实现十进制小数的转换规则是：用 2 连续乘以要转换的十进制数及各次所得之积的小数部分，一直乘到积的小数部分为 0 时止，则各次所得之积的整数部分即为所求十进制数由高位到低位的值。

需指出的是，在用上述规则实现十进制小数到二进制小数转换时，会出现积的小数部分总不等于 0 的情况，这表明此时的十进制小数不能转换为有限位的二进制小数，即出现了“循环小数”，如

$$(0.6)_{10}=(0.100110011001\cdots)_2$$

在这种情况下，乘 2 取整是否结束由所要求的转换位数（即转换精度）决定。

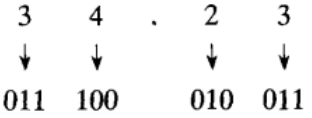
当十进数包含有整数和小数两部分时，可按上面介绍的两种方法将整数和小数分别转换，然后相加。

(3) 二进制数与八进制数的转换

由于八进制数的基数为 8，二进制数的基数为 2，两者满足 $8=2^3$，故每位八进制数可转换为等值的三位二进制数；反之亦然。

[例 1-4]　计算 $(34.23)_8=(?)_2$。

只要将每位八进制数写成等值的三位二进制数，便可得到转换的结果，即



求得 $(34.23)_8=(11100.010011)_2$

[例 1-5] 计算 $(11110100.10111)_2=(?)_8$。

以小数点为界，整数部分从右到左分成三位一组，小数部分从左到右分成三位一组，头尾不足三位时补 0，再将每组的三位二进制数写成一位八进制数，则得

011 110 100 . 101 110
↓ ↓ ↓ ↓ ↓
3 6 4 5 6

求得 $(11110100.10111)_2=(364.56)_8$

(4) 二进制数与十六进制数的转换

由于十六进制数的基数为 16，二进制数的基数为 2，两者满足 $16=2^4$，故每位十六进制数可转换为四位二进制数；反之亦然。

[例 1-6] 计算 $(D8.C4)_{16}=(?)_2$。

将每位十六进制数写成四位二进制数，便得转换的结果，即

D 8 . C 4
↓ ↓ ↓ ↓
1101 1000 1100 0100

求得 $(D8.C4)_{16}=(11011000.110001)_2$

[例 1-7] 计算 $(11011110.10101)_2=(?)_{16}$。

以小数点为界，整数部分从右到左分成四位一组，小数部分从左到右分成四位一组，头尾不足四位时补 0，然后将每组的四位二进制数写成一位十六进制数，即

1101 1110 . 1010 1000
↓ ↓ ↓ ↓
D E A 8

求得 $(11011110.10101)_2=(DE.A8)_{16}$

从例 1-4～例 1-7 可以看出，二进制数与八进制数或十六进制数之间存在直接转换关系。可以说，八进制数或十六进制数是二进制数的缩写形式。在计算机中，利用这一特点可以把用二进制数代码表示的指令或数据写成八进制数或十六进制数的形式，以便于书写或认读。

1.3.2 计算机中数的表示

在计算机中，表示小数点的方法有两种：一种是定点表示法，另一种是浮点表示法。

1. 定点表示法

所谓定点（fixed point）表示法，指的是计算机中的小数点位置固定不变。根据小数点位置的固定方法不同，定点表示法可分为定点整数表示法和定点小数表示法。前者小数点固定在数的最低位之后，后者小数点固定在数的最高位之前。设计算机的字长为 8 位，则上述两种表示法的格式如图 1-2 所示。

显然，对于定点整数表示法而言，它能表示的数值范围（绝对值）为

00000001～01111111

即

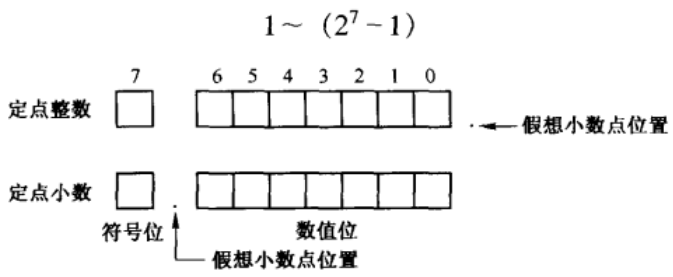


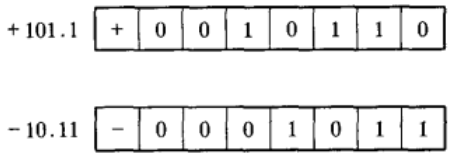
图 1-2　定点表示法

显而易见，计算机采用定点整数表示时，它只能表示整数。但在实际问题中，数不可能总是整数，这就需要在用计算机解题之前对非整数进行必要的加工，以变为适合于计算机表示的形式。+101.1 和 −10.11 都是非整数，若将它们都乘以比例因子 2^2，则得

$$+101.1\times 2^2 = +10110$$

$$-10.11\times 2^2 = -1011$$

在 8 位字长的计算机中可分别表示为



当然，在输出结果时必须除以比例因子。

类似地，定点小数表示法只能表示小于 1 的数，与定点整数表示法无原则区别。可参照定点整数表示法自行分析。

2. 浮点表示法

所谓浮点（floating point）表示法，是指计算机中的小数点位置不是固定的，或者说是“浮动”的。为了说明它是怎样浮动的，我们引入二进制数的另一种表示法——记阶表示法。对于任何一个二进制数 N，都可表示为

$$N=2^{\pm E}\times(\pm S)$$

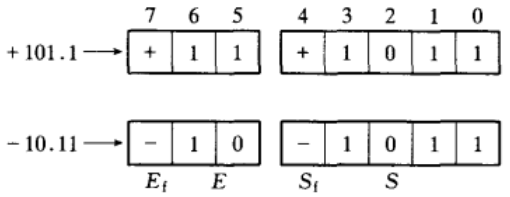
式中，E 为阶码，是一个二进制数；E 前的“±”号为阶码的正负号，称其为阶符（E_f）；S 为尾数，是一个二进制正小数；S 前的“±”号为尾数的正负号，称其为尾符（S_f）；“2”为阶码 E 的底数。在某些中、大型计算机中，阶码的底数可为“16”。

例如，二进制数 +101.1 和 −10.11 的记阶表示形式为

$$+101.1=2^{+11}\times(+0.1011)\quad\begin{cases}E=11 & E_f\text{为“+”}\\ S=0.1011 & S_f\text{为“+”}\end{cases}$$

$$-10.11=2^{+10}\times(-0.1011)\quad\begin{cases}E=10 & E_f\text{为“+”}\\ S=0.1011 & S_f\text{为“−”}\end{cases}$$

采用记阶表示法后，则计算机只需表示出它的阶码、尾数及其符号，阶码的底数“2”可以不表示出来。若用 8 位字长中的两位表示阶码，4 位表示尾数，另两位分别表示阶符和尾符，则上述两个二进制数在计算机内的浮点表示形式为



比较+101.1和-10.11两数可知，它们的有效数字（1011）是完全相同的，只是正、负号和小数点的位置不同。正、负号的不同反映在尾符 S_f不同。因此，对于规格化的浮点数（其尾数 $1/2 \leqslant S < 1$）而言，小数点的位置是随阶码的大小而“浮动”的。

3. 二进制数的原码、反码及补码的表示

在计算机中，数的正、负号是用“0”、“1”表示的。

例如，二进制数+101.1和-10.11可表示为

取比例因子 2^2

$$+101.1 \longrightarrow +0010110 \longrightarrow 0,0010110$$
$$-10.11 \longrightarrow -0001011 \longrightarrow 1,0001011$$

在该例中，自左至右的第一式（+101.1和-10.11）是给定的二进制数；第二式（+0010110和-0001011）是乘比例因子后的二进制整数形式，它除符号（“+”或“-”）外已能在计算机中表示（设该计算机采用定点整数表示），则称该数为计算机的真值；第三式（0，0010110和1，0001011）是真值符号“数值化”后所得的数，它已完全能在计算机中表示，故称其为机器数。机器数的最高位为符号位，为明显起见，常用逗号“，”隔开（也可以省略）。

一般地，通过对机器数的定义，可以将真值的符号“数值化”。下面仅讨论常用机器数——原码（primary code）、反码（one's complement）和补码（two's complement）是如何由真值得到的。

设计算机的字长为 n 位，它可表示的真值 $x = \pm x_{n-2}x_{n-3}\cdots x_0$。其中，$x_i = 0$ 或 1，则有

- 真值 $x = +x_{n-2}x_{n-3}\cdots x_0$时，原码、反码和补码完全相同，即

 $[x]_{原} = [x]_{反} = [x]_{补} = 0x_{n-2}x_{n-3}\cdots x_0$

- 真值 $x = -x_{n-2}x_{n-3}\cdots x_0$时，原码、反码、补码与 x 的关系为

 $[x]_{原} = 1\ x_{n-2}x_{n-3}\cdots x_0$

 $[x]_{反} = 1\ \overline{x}_{n-2}\overline{x}_{n-3}\cdots\overline{x_0}$

 $[x]_{补} = 1\ \overline{x}_{n-2}\overline{x}_{n-3}\cdots\overline{x_0} + 1$

由上可知，在 n 位的机器数中，最高位为符号位。该位为0，表示真值为正；为1，表示真值为负；其余（$n-1$）位为数值位，各位的值可为0或1。当真值为正时，原码、反码和补码的数值位完全相同；当真值为负时，原码的数值位保持真值的原样，反码的数值位为原码的各位取反，补码则是反码的最低位加1。

根据上述关系，很容易实现真值与机器数之间及三种机器数之间的相互转换。下面将举例说明。

[例 1-8] 已知计算机字长为8位，试写出二进制数+101010和-101010在机器中表示

的原码、反码及补码。

先写出这两个二进制数的真值。若该机器采用定点整数表示，则其真值形式为

$$x = +0101010$$

$$y = -0101010$$

即连同符号构成 8 位（字长）。为使所表示的值保持不变，故在不足 7 位的数值位前添 0。按真值与机器数之间的关系，写出机器数。

真值 x 为正，则有

$$[x]_{原} = [x]_{反} = [x]_{补} = 00101010$$

真值 y 为负，则有

$$[y]_{原} = 10101010$$

$$[y]_{反} = 11010101$$

$$[y]_{补} = 11010110$$

[例 1-9] 已知 $[x]_{补} = 101101$，求真值 x。

先由 $[x]_{补}$ 求出 $[x]_{反}$，则得

$$[x]_{反} = 101101 - 1 = 101100$$

$[x]_{反}$ 的符号位为 1，故其所对应的真值为负，且数值位为 $[x]_{反}$ 的各位取反，即

$$[x]_{反} = 101100$$

$$x = -10011$$

在计算机中，参与运算的是机器数，不同机器数的运算规则的复杂程度不同。我们期望所选用的机器数不仅与真值之间的转换关系直观简便（如原码），而且运算规则简单（如补码的加减运算）。

1.3.3　计算机中的编码

所谓编码（code），就是用按一定规则组合而成的若干位二进制码来表示数或字符（字母、符号和汉字）。这是计算机中所采用的按“形”表示数的一种方法。计算机中常用的编码有十进制编码、可靠性编码、字符编码、操作编码及汉字编码等。不同的编码，其编码规则不同，且具有不同的特性及应用场合。

1. 十进制编码

十进制编码是指用若干位二进制代码来表示一位十进制数。其中，最常用的是 BCD（Binary Coded Decimal）码。它用四位权为 8421 的二进制数来表示等值的一位十进制数。其编码规则见表 1-3。按表 1-3 给定的规则可以很容易实现十进制数与 BCD 码之间的转换。

表 1-3　BCD 码与其奇偶校验码

十进制数	BCD 码	BCD 校验码	十进制数	BCD 码	BCD 校验码
0	0000	00001	5	0101	01011
1	0001	00010	6	0110	01101
2	0010	00100	7	0111	01110
3	0011	00111	8	1000	10000
4	0100	01000	9	1001	10011

2. 可靠性编码

计算机进行数据传输或存取时免不了要出错。为了能及时地发现错误，并及时地检测与校正错误，计算机采用了可靠性编码。常用的可靠性编码有格雷码（Gray code）、奇偶校验码（Odd even check code）和海明码（Hamming code）等。

在格雷码中，任意两个相邻代码只有一位二进制数不同，因而当数据做顺序改变时不会发生“粗大”误差，从而提高了可靠性。海明码则是一种既能检测出错位也能校正出错位的可靠性代码。奇偶校验码是一种被广泛采用的可靠性编码。它由若干信息位加一个校验位组成。其中，校验位的取值（0 或 1）将使整个代码中“1”的个数为奇数或偶数。若“1”的个数为奇数，则称奇校验码；否则，称偶校验码。

奇偶校验码具有检测一位错的能力。例如，若约定计算机中的二进制代码都是以奇校验码存入存储器的，那么当从存储器取出时，若检测到某一二进制代码中“1”的个数不是奇数，则表明该代码在存取过程中出现了错误，但不知是哪一位错，因此奇偶校验码无自动校正能力。不难理解，若代码在存取过程中发生了两位错，则用奇偶校验就检测不出来。

3. 字符编码

计算机中的信息包括数据信息和控制信息。数据信息又可分为数值信息和非数值信息。非数值信息和控制信息中包括了字母、各种控制符号及图形符号等。它们都以二进制编码方式存入计算机并得以处理。这种对字母和符号进行编码的二进制代码称为字符代码（character code）。

在计算机中，常用的字符编码有 ASCII 码（美国标准信息交换代码）和 EBDIC 码（扩展的 BCD 交换代码）。表 1-4 给出了 ASCII 码的字符集。由表 1-4 不难写出 SIN（3.14/N）的 ASCII 码为

S I N (3 . 1 4 / N)

1010011 1001001 1001110 0101001 0110011 0101110 0110001 0110100 0101111 1001110 010100

为书写方便，常把 ASCII 码的 7 位二进制代码写成两位十六进制数。例如，上例中“S”的 ASCII 码为 53H，“I”的 ASCII 码为 49H。依此类推。

表 1-4 ASCII 码的字符集

低 4 位 \ 高 3 位	000	001	010	011	100	101	110	111
0000	NULL	DC1	空格	0	@	P	、	p
0001	SUM	DC2	!	1	A	Q	a	q
0010	EOA	DC3	"	2	B	R	b	r
0011	EOM	DC4	#	3	C	S	c	s
0100	EOT	DC5	$	4	D	T	d	t
0101	WRU	ERR	%	5	E	U	e	u
0110	RU	SYNC	&	6	F	V	f	v
0111	BELL	LEM	、	7	G	W	g	w
1000	BKSP	S	)	8	H	X	h	x
1001	HT	S	(	9	I	Y	i	y

续表

低 4 位 \ 高 3 位	000	001	010	011	100	101	110	111
1010	LF	S	*	:	J	Z	j	z
1011	VT	S	+	:	K	[	k	\|
1100	FF	S	,	<	L	\	l	\|
1101	CR	S	-	=	M	]	m	\|
1110	SO	S	.	>	N	^	n	ESC
1111	SI	S	/	?	O	-	o	DEL

4. 汉字编码

在计算机中，为了解决汉字的输入、处理及输出问题，出现了各种汉字编码（chinese character code）方案，包括汉字的输入编码、机内编码及汉字点阵图形。

用键盘输入汉字时所使用的汉字编码称作输入码。它与汉字输入方式有关，常见的有汉语拼音码、五笔字型码、音形码、国标区位码及电报明码等数十种之多。每种输入码各有特点，如汉语拼音码是以字音为输入依据，使用方便，平均输入一个字需击 3～4 次键，但重码较多。五笔字型是以字形为输入依据，将汉字分为五种字根，用专用键帽指示，易学易用，熟练后输入汉字迅速。

不管以何种编码方式输入汉字，最终都要转换为机内代码，以实现汉字在计算机内的储存与处理。目前，汉字机内码统一使用国家标准 GB 2312—80 规定的国际区位码。它收集了汉字和其他语言图形符号 7445 个，其中 6763 个为汉字。国标区位码中的编码按其位置分为 94 区，每区有 94 个字符，故“区位码”由此得名。区位码中的每个汉字用其所在的“区”、“位”号进行编码，并用两个字节（8 位二进制）分别表示区号和位号，每个字节的最高位置“1”用作汉字标记。

[例 1-10]　汉字“啊”的区位码为 3021H，用两个最高位置“1”的字节表示该码时，所得的机内码实际为 B0A1H，如图 1-3 所示。

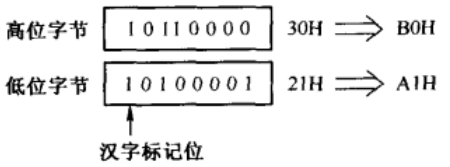


图 1-3　“啊”的机内码

汉字在显示和打印时均以点阵方式输出，常用 16×16 点阵表示一个汉字，为此需在计算机内（或 CRT 显示器、打印机内）建立汉字库。该库内以图形方式存储了国标所规定的两级汉字。当需要输出汉字时，计算机便根据汉字的机内码从汉字库中取出相应的汉字点阵图形，显示在 CRT 上或从点阵式打印机上打印出来。汉字库除采用 16×16 点阵外，还有压缩型的 8×16 或 16×8 点阵，以及扩展型的 32×32、48×48、128×128 点阵，可输出几十种字体的大小汉字。

小结

本章主要介绍了计算机发展简史、计算机的应用和发展趋势、计算机系统的组成和工作原理、数制与编码等几方面的知识。

关于计算机的发展简史，主要应该知道计算机的发展阶段是以计算机元器件的变革作为划分的标志，并了解当前计算机采用的元器件是什么，了解计算机的应用领域和计算机的发展趋势。

计算机系统是由硬件系统和软件系统两部分组成的。关于计算机硬件系统，主要应掌握计算机硬件的五大组成部分、各部分的作用及计算机的基本工件原理；关于计算机软件系统，主要应掌握计算机软件的分类和各类软件的作用。

数据与编码部分应掌握进制数及其相互转换、数的表示和编码三个方面的知识。数制相互转换方面主要应掌握二进制数与八进制数、十进制数、十六进制数的相互转换方法；数的表示方面主要应掌握计算机中的数的定点表示方法和浮点表示方法，掌握进制数的原码、反码和补码的表示方法。编码方面主要应了解计算机中非数值型数据的表示方法及汉字在计算机中的表示方法。

习题

1. 计算机的主要特点有哪些？计算机的发展主要分为四个阶段，划分的依据是什么？
2. 试述计算机应用系统的发展趋势。
3. 试述计算机的基本组成及各组成部分的功能。
4. 简述计算机的基本工作原理。
5. 设 $x=(1110)_2$，$y=(1011)_2$，求 $x-y$，$x+y$。
6. 完成下列数制的转换：

 $(121)_{10}=(\quad)_2=(\quad)_8=(\quad)_{16}$

 $(0.625)_{10}=(\quad)_2=(\quad)_8=(\quad)_{16}$

 $(63.43)_{10}=(\quad)_2=(\quad)_8=(\quad)_{16}$

 $(FF)_{16}=(\quad)_8=(\quad)_2$

 $(B9)_{16}=(\quad)_8=(\quad)_2$
7. 完成下列数制转换：

 $(1111010101.110101)_2=(\quad)_8=(\quad)_{16}$

 $(11010010.11)_2=(\quad)_8=(\quad)_{16}$

 $(1010101111.1)_2=(\quad)_8=(\quad)_{16}$

 $(BA.FE)_{16}=(\quad)_8=(\quad)_2$
8. 已知二进制数 $x_1=+0.10111$，$x_2=-0.101011$，$x_3=+1011101$，$x_4=-1011011$，求 x_1，x_2，x_3，x_4 的原码、补码和反码。

第 2 章　计算机硬件基础

本章要点

- ☑ 中央处理器：运算器、控制器
- ☑ 存储器：半导体存储器、磁介质存储器、光介质存储器
- ☑ 输入设备：键盘、鼠标、光学标记阅读机、图形扫描仪
- ☑ 输出设备：显示器、打印机、绘图仪
- ☑ 输入输出设备接口

一个完整的计算机系统通常是由硬件系统和软件系统两大部分组成的。可以这样认为，硬件系统是计算机系统的身躯。软件系统是计算机系统的头脑和灵魂。只有这两者紧密地结合在一起，才能成为有生命、有活力的计算机系统。硬件系统和软件系统是互相依存、密不可分的。

2.1　中央处理器

中央处理器的英文名称是 Control Processing Unit，简称为 CPU。它是由运算器和控制器两部分组成的。

2.1.1　运算器

运算器主要由算术逻辑单元（ALU）、寄存器及一些数据传送的电路组成。算术逻辑单元是运算器中实现算术逻辑运算的电路。寄存器是运算器中的数据暂存器。运算器中往往设置多个寄存器。每个寄存器都能够保存一个数据。在运算过程中，运算器不断地得到由存储器或寄存器提供参加运算的数据，运算的中间结果可以保存在累加寄存器（简称累加器，Accumulator）中或送回存储器中。如图 2-1 所示，运算器整个运算过程是在控制器统一指挥下，按程序中编排的操作次序进行的。由寄存器提供数据，中间结果暂存寄存器中，使运算过程就可以在运算器内部完成，不需要频繁地与存储器打交道，从而提高了运算速度。

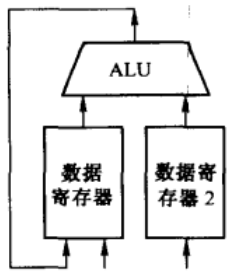


图 2-1　运算器运算过程示意图

以 A + B 为例，先将数 A 放到累加器中，再把数 B 取到寄存器中。此时，数据 A、B 在控制器的控制下，就可以送到算术逻辑单元中进行加法运算。其相加的结果被送到累加器，等待下次的运算或按程序指令送回存储器中保存起来。

实际上，四则运算加、减、乘、除等算法都可归结为加法与移位操作，所以加法器是算

术逻辑运算电路中的关键部件。

2.1.2 控制器

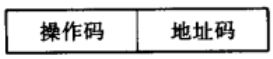
计算机的工作过程是程序的运行过程，也就是在控制器的控制下逐条执行程序中指令的过程。控制器是计算机的指挥中心和控制中心，决定整个计算机在什么时间、根据什么条件做什么事。

1. 控制器的功能

控制器最基本的功能：①按照程序的要求，控制指令执行的顺序；②控制每一条指令的具体执行，即控制器根据指令计数器 PC 中保存着的要执行的下一条指令的地址，从内存储器中取出指令，并送到 IR 中，经过指令译码器 ID 对 IR 中的指令代码进行分析、翻译后，发出各种相应的操作命令，指挥计算机的有关部件进行工作。

2. 指令格式

所谓指令，是指计算机硬件可执行的且可完成一个基本操作的命令。指令明确告诉计算机硬件控制线路，即从存储器的哪个单元取数，在运算器中进行何种操作。也就是说，指令需要由两部分组成，即操作码和地址码。操作码决定指令的功能；地址码则指示从存储器哪个存储单元中把要运算的数据取出来，再送到运算器中，进行操作码所指定的运算。一条指令至少包含两个部分，一般格式为



指令随计算机的不同而异，也可能有许多附加的条件。不同类型的计算机，由于其硬件结构不同，故指令也不同，但最基本的格式是大致相同的。某一个计算机的基本指令的集合，称为该计算机的指令系统。

计算机的指令系统至少应具备的基本指令有：数据传送指令、算术逻辑指令、程序控制指令和输入输出指令。

一台计算机指令的数量可能有多有少，少则 8～16 条，多则几百条。指令的条数和指令的功能是一台计算机功能的主要标志之一。

3. 控制器的组成

控制器概括起来可以分为以下几个部分，如图 2-2 所示。

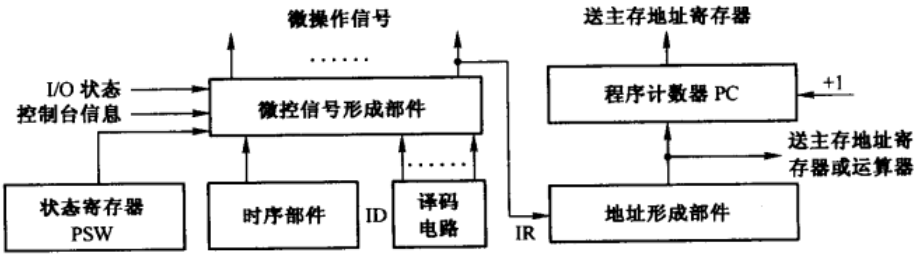


图 2-2 控制器基本组成框图

(1) 指令部件

- 程序计数器 PC：用来存放现行指令的地址。每当执行完一条指令后，通常由 PC 提供后继指令地址，送往主存储器的地址寄存器。计数可由逻辑电路实现，也可经运算器来实现。
- 指令寄存器 IR：专门用来存放现行指令，一直保存到指令译码器译码完成为止。
- 指令译码器 ID：对现行指令进行分析，输出相应的操作控制电位。
- 程序状态寄存器 PSW：用来存放程序的工作方式和运算结果。其内容被称为程序状态字（PSW）。程序状态字表明了系统的基本状态，是参与控制程序执行的重要依据之一。

(2) 时序部件

时序部件用来产生计算机各部件所需的定时信号。

(3) 控制信号形成部件

控制信号形成部件用来产生所有指令所需的各种微操作控制信号。

4. 指令的执行

一条指令的执行主要分为两个阶段。

- 从存储器中取出指令。根据 PC 的内容，向存储器地址寄存器发出地址和读信号，取出现行指令后，存放到控制器中专门设置的 IR 中。
- 分析和执行。分析指令就是把 IR 中的当前指令进行译码，即首先分析其操作码，为此需设置操作码译码器，通过对操作码的分析，产生相应的控制电位，并提供给控制信号形成部件。分析指令的另一个内容就是确定参与操作的数据地址，即分析寻址方式和形成操作数地址的地址码，并向存储器发出调用操作数的控制信号，取得操作数后，根据指令的操作性质和 CPU 的某些状态特征，执行指令所给定的操作。

2.2　存储器

计算机系统中用来记录信息的设备称为存储器。信息是指指令和数据。它们都是以二进制数表示的，所以存储器中存放的是二进制数 0 和 1。

2.2.1　存储器概述

1. 基本概念

存储器是保存信息的场所，在计算机中有着十分重要的地位。存储器的容量越大，表明能容纳的信息越多，把信息存入存储器或从存储器中取出信息的速度越快，计算机处理信息的速度就越快。

存储器由一些能表示二进制数“0”和“1”的物理器件组成。称这种器件为记忆元件或记忆单元。常用的存储介质有半导体器件和磁性材料。每一个记忆元件可以存储一位二进制代码信息（即 0 或 1）。位是二进数的基本单位，也是存储器中存储信息的最小单元。由 8 位组成的二进制代码称为一个字节。由一个或多个字节组成的，并作为一个整体进行存取的数为一个字。大量存储单元的集合组成一存储体。我们可以将存储器视为一个存放信息的大

仓库，而一个仓库又可以分为若干个小房间，把每个房间称为一个单元，每个单元的编号被称为地址。若一个单元存放一个字节，则相应的地址被称为字节地址；一个单元存放一个字，则相应的地址被称为字地址。

一个存储器中存储单元的总和被称为存储器的存储容量。表示存储器容量的单位一般为字节（B），通常用千字节、兆字节或吉字节表示。2^{10}个字节被称为 1 千字节，记作 1 KB，即1 KB＝1024 B，而不是 1000 B；2^{20}个字节被称为 1 兆字节，记作 1 MB，即 1 MB＝1024×1024＝1 048 576 B；2^{30}个字节被称为 1 吉字节，记作 1 GB，即 1 GB＝1024×1024×1024＝1 073 741 824 B。例如，内存容量为 640 KB，就是说有 640×1024 个字节，含有 8×640×1024 个二进制位。

计算机内存容量的大小决定了计算机处理问题能力的大小。

存储器的两个基本操作是读出与写入。读出操作是把存储单元内的信息取出。写入操作是把信息存入存储单元内。执行这两种操作都需要一定的时间。把数据总线上的信息写入存储器的时间称为写入时间。从存储器中存取一个字到能够再存取下一个字所需要的时间称为存取周期。存取周期表示了存储器的工作速度。存取周期越短，存储器的工作速度越快。

2. 存储器的分类

由于信息载体和电子元器件的不断发展，存储器的功能和结构都发生了很大变化，故出现了各种类型的存储器，以适应计算机系统的需要。下面简单介绍一下存储器的各种分类方法。

（1）按功能和存取速度分类

根据功能和存取速度，存储器可分为寄存器型存储器、高速缓冲存储器、内存储器及外存储器。

寄存器型存储器是由多个寄存器组成的存储器，如当前许多 CPU 内部的寄存器组。它可以由几个或几十个寄存器组成。其字长与计算机字长相同，主要用来存放地址、数据及运算的中间结果，速度可与 CPU 匹配，但容量极小。

高速缓冲存储器是计算机中的一个高速小容量存储器。其中存放的是 CPU 近期要执行的指令和数据。一般采用双极型半导体存储器作为高速缓冲存储器。

内存储器安装在主机内部，直接与 CPU 交换信息。内存储器是计算机存放在运行期间较常用的大量程序和数据的主要存储器，故又称为主存储器。内存储器一般由半导体存储器构成，包括随机读写存储器和只读存储器两个部分。目前使用的计算机，其内存容量在几十兆到几百兆之间。

外存储器是安装在计算机外部的存储器，也叫辅助存储器。它的容量很大，但存取速度较低，主要由磁表面存储器组成，如目前广泛使用的磁盘存储器和光盘存储器，主要用来存放当前暂不参加运算的程序和数据。

（2）按存取方式分类

按存取的方式，存储器可分为随机读写存储器、只读存储器和顺序存取存储器。

随机读写存储器（Random Access Memory，RAM）是一种通过指令可以随机读出和写入存储器中任一单元内信息的存储器。RAM 的存取时间基本固定，与 CPU 执行指令的速度相配，通常用来存放用户的程序和数据等。计算机内存大部分由 RAM 组成。

值得注意的是，RAM 只是临时存储信息，一旦断电，RAM 中的信息便立即丢失。

只读存储器（Read Only Memory，ROM）在一般情况下只能从存储器内读取信息，不能写入信息，要写入信息需借助专门的仪器。ROM 一般存放系统配置的若干软件和用户开发的专用程序，通常由厂家将磁盘引导程序、自检测程序及 I/O 驱动程序等写入 ROM 中，可避免被破坏。ROM 与 RAM 的重要区别：信息一旦被写入后，就被固定下来，即使断电也不会丢失，因此可以长久地保存信息。

顺序存取存储器（Sequential Access Memory，SAM）是一种只能按顺序来存取信息的存储器，如磁带存储器。通常 SAM 的存储周期较随机读写存储器长。

(3) 按存储介质分类

按存储介质，存储器可分为磁性材料存储器、半导体存储器及光介质存储器。

磁性材料存储器又称为磁存储器，包括磁心存储器、磁盘存储器、磁带存储器及磁鼓存储器等。其中，磁盘存储器、磁带存储器和磁鼓存储器又被统称为磁表面存储器。它们都是以磁性材料为存储介质的。现在，磁心存储器和磁鼓存储器已被淘汰。计算机中广泛使用的外存是磁盘，有软盘和硬盘之分，能长期存储大容量的信息。

半导体存储器是一种用半导体材料组成的存储器，目前有两大类：一种是双极型，又分为 TTL 型和 ECL 型两种；另一种是金属氧化物半导体存储器，简称为 MOS 存储器，又有静态 MOS 存储器和动态 MOS 存储器之分。它的存取速度快，用来构成主存，与 CPU 直接交换信息。

光介质存储器是一种类似激光唱盘的存储器。信息以刻痕的形式保存在盘面上，用激光束照射盘面，靠盘面的不同反射来读出信息。正因为如此，人们常称这类存储器为光盘。光盘可分为只读型光盘、可刻录光盘和可改写型光盘 3 种。

2.2.2　半导体存储器

半导体存储器从使用功能上划分可分为两类，即随机读写存储器 RAM 和只读存储器 ROM。RAM 主要用来存放各种现场的输入输出数据、中间计算结果、与外存交换的信息及作为堆栈使用。其存储单元的内容可按照需要被读出，也可以写入或改写。而 ROM 的信息在使用时是不能改变的，也就是不可写入的，只能读出，故一般用来存放固定的程序（如计算机的管理程序、监控程序及汇编程序等）及存放各种常数、函数表等。

1. 随机存储器

(1) RAM

RAM 的全名是读写随机存取存储器（Read Write Random Access Memory），本应缩写为 RWRAM，但它不易发音，故流行写为 RAM。

RAM 通常有三个特点：

- 可以读出，也可以写入。读出并不损坏所存储的内容，只有写入时才可修改原来所存储的内容。
- 所谓随机存取，意味着存取任一单元所需要的时间相同。因为存储单元被排成二维队列，就像通过 x、y 两坐标就能确定一个点那样，可以通过行地址和列地址来存取一个单元。
- 当断电后，存储内容立即消失，称其为易失性。不过，今天已有非易失性 RAM，被称为 NOVRAM（nonvolatile RAM）或 NVRAM。

(2) DRAM 与 SRAM

RAM 可分为动态 RAM 和静态 RAM 两大类。

所谓动态 RAM (Dynamic RAM)，是指用 MOS 电路和电容作为存储元器件。由于电容存在泄漏现象，所以需要定时补充电荷以维持存储内容的正确。这一过程被称为“刷新”。例如，每隔 2 ms 刷新一次，因此称为 DRAM。

所谓静态 RAM (Static RAM)，是指用双极型电路或 MOS 电路的触发器作为存储元器件，没有电容造成的放电问题。只要有电源正常供电，触发器就能稳定地存储数据，因此称其为 SRAM。

DRAM 的特点是高密度；SRAM 的特点是高速度。

自 1970 年 Intel 公司推出 1 KB DRAM 以来，半导体存储器走上迅猛发展的道路。DRAM 一直是存储器市场的主流产品。由于 DRAM 具有高密度的特点，所以集成度的提高非常迅速。集成度差不多以每三年增加四倍的速度发展着。与此同时，DRAM 的性能也在不断地提高。

(3) NVRAM

NVRAM 是一种非易失性的 RAM，即能快速存取，在系统断电时又不丢失数据。实际上，它是将 SRAM 的实时读写功能与电可擦除可编程 ROM 的可靠非易失能力综合在一起的。

以 Intel 2004 NVRAM 为例，它是一块 4 KB 芯片，以 512×8 “字节宽”体系结构组成。其内部结构分为两部分：一部分是高速静态 RAM 阵列；另一部分是与它逐位对应的非易失电可擦除可编程 ROM 备份阵列。

系统正常工作时，CPU 访问 SRAM 部分以完成快速读写。当系统断电时，或者正常关机时，芯片内部的数据保护电路在测出电源降至 4V 时能立即关闭写入电路，而迅速将 SRAM 的内容并行转储到电可擦除可编程 ROM 中。电源恢复后，电可擦除可编程 ROM 中的内容又自动放入 SRAM 中。这种转储操作能可靠地进行 1 万次，非易失能力保证能存储 10 年以上。

2. 只读存储器

只读存储器 (Read Only Memory) 按首字母可缩写为 ROM。其特点是只能读出原有内容，而不能写入新内容。ROM 按存储单元的结构和生产工艺的不同，可构成下面几种 ROM 存储器。

(1) 固定掩膜 ROM

固定掩膜 ROM 的每一个存储单元由单管构成，因此集成度较高。存储单元的编程是在生产过程中，由厂家通过掩膜这道工序将信息做到芯片里，也就是将单管电极接入电路，未接入电路的位存 1；否则，存 0。

固定掩膜 ROM 的编程 (信息的写入) 只能由器件制造厂在生产时定型。若要修改，则只能在生产厂重新定做新的掩膜，而用户无法自己操作编程。

(2) 可编程 ROM (PROM)

可编程 ROM (Programmable ROM，PROM) 允许用户自己编程一次。在 PROM 中，常采用二极管或双极型三极管做存储单元。管子的发射极上串接可熔性金属丝。该熔丝的完好与否，决定该位信息的状态。出厂时，所有熔丝是完整的，管子将位线与字选线连通，表示

存有 0 信息，因此新出厂的 PROM 芯片应为全 0 状态。用户编程时，在脉冲的作用下，使熔丝断开，该位由 0 变为 1 状态，实现了信息的写入。用户只要控制该往哪位写 1，便实现了对 PROM 的编程。由于熔断丝烧断后无法恢复，所以 PROM 芯片只能进行一次编程。

(3) 紫外光擦除可编程 ROM（EPROM）

紫外光擦除可编程 ROM（Erasable Programmable ROM，EPROM）芯片常用浮栅型 MOS 管做存储单元。新出厂的“干净”EPROM 每位均为 1 状态。对 EPROM 的编程是用电信号控制将有关位由原来的 1 改写为 0 的过程。对 EPROM 的擦除过程则是用紫外光照射，即用高能光子将浮栅上的电子驱逐出去，使其返回基片，相应位由原来的 0 变为 1 状态。由于紫外光通过 EPROM 的石英窗口对整个芯片的所有单元都发生作用，所以一次擦除便可使整个芯片恢复为全 1 状态，部分擦除是不行的。

对 EPROM 的擦除和写入都有专用设备。写入之前应确保芯片是“干净”的，即为全 1 状态。EPROM 的写入器一般可对多种型号的 EPROM 芯片进行写入，并通过读写芯片的识别码来确认该使用什么样的编程脉冲和编程电压。

(4) 电可擦除可编程 ROM（EEPROM）

电可擦除可编程 ROM（Electrically Erasable Programmable ROM，EEPROM 或 E^2PROM）是一种不用从电路上插上、拔下，而直接在线用电信号进行擦除的 EPROM 芯片。对其进行的编程也是在线操作，因此它的改写步骤简单。其他性能与 EPROM 类似。

(5) 闪速存储器

闪速存储器是一种新型的半导体存储器。由于它具有可靠的非易失性、电擦除性及低成本等优点，故对于需要实施代码或数据更新的嵌入性应用是一种理想的存储器，而且它在固有性能和成本方面有较明显的优势。

Intel 公司的 ETOX 闪速存储器是以单晶体管 EPROM 的存储单元为基础的，因此闪速存储器就具有非易失性，在断电时它也能保存存储的内容，这使它优于需要持续供电来存储信息的易失性存储器，如静态和动态 RAM。闪速存储器的存储单元结构和它具有的EPROM 基本特性使它的制造特别经济，在存储密度增加时能保持可测性，并具有可靠性。这几方面的优势综合起来是目前其他半导体存储器技术所无法比拟的。

与 EPROM 只能通过紫外线照射实施擦除的特点不同，闪速存储器可实现大规模电擦除。闪速存储器的擦除功能可迅速清除整个存储器中的所有内容。这一点优于传统的可修改的 E^2PROM。闪速存储器可重复使用，可以被擦除和重新编程几十万次而不会失效。在文件需经常更新的可重复编程应用中，这一性能是非常重要的。

总之，闪速存储器是一种低成本、高可靠性的读写非易失性存储器。从功能上讲，由于随机存取的特点，闪速存储器也可看作是一种非易失的 ROM，因此它成为能够用于程序代码的数据存储的理想媒体。

2.2.3　磁介质存储器

1. 磁表面存储器的存储原理

磁表面存储器是用某些磁性材料涂在金属铝片或塑料片（带）的表面上作为载磁体而存储信息的存储器。其存储信息的原理可用如图 2-3 所示来说明。

在载磁体的附近有一个磁头，其上绕有一个读写线圈。写操作时，在磁头的线圈中通以一定方向的脉冲电流，磁头的铁心内便产生一定方向的磁通。该磁通使磁头空隙下的载磁体局部磁化，形成相应极性的磁化元。写入脉冲电流的方向不同，在载磁体上形成的磁化元的极性便不同。用一种极性表示信息“1”，另一种极性表示信息“0”，则一个磁化元就是一个存储元件，可存储一位二进制信息。可见，写操作过程是将两个方向的电流脉冲转化为两种极性磁化元的过程。该磁化元的极性可永远地保存在载磁体上，直到下一次重新写入为止。

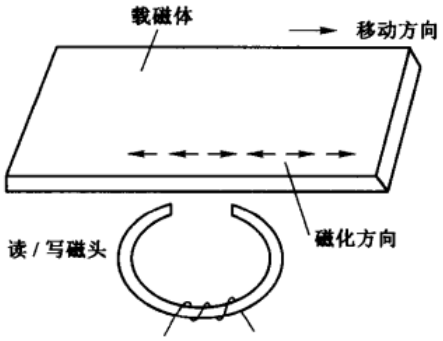


图 2-3 磁表面存储器存储信息的原理

读操作与此相反，是将不同极性的磁化元状态转化为不同方向的电流脉冲的过程。其原理是：当载磁体按一定方向移动时，磁头做切割磁力线运动，便在磁头的线圈中产生感应电势。电势在外电路的作用下转化为脉冲电流。

2. 磁带存储器

磁带存储器类似于家用的磁带录音机。录音机记录的是模拟信息，而磁带存储器记录的是数字信息。磁带机由磁带、磁盘、读写磁头、主导轮、驱动磁带盘转动的电机及控制电路等组成。磁带是一种表面涂有磁性材料的塑料带，绕在磁带盘上。当磁盘在电机的驱动下转动时，磁带从一个盘卷向另一个盘，使磁盘在读写磁头下移动。在 CPU 发出的指令控制下，通过读写磁头便可实现对磁带的读写操作。为了能快速启动，在磁头的两边各留有一段自由带，作为加速时的缓冲区，以减轻带盘惯性的影响。

磁带机是一种顺序存取的存储器。磁带上的信息以信息块的形式（一个“记录”或一个“文件”）顺序地存放在磁带上。其存放格式如图 2-4 所示。各信息块之间留有间隙，每个信息块都有“头标”和“尾标”。若磁头当前的位置在磁带首部，而要读取的信息块在磁带的尾部，则必须空转磁带到尾部，才能读取该信息块。因此，磁带的存取时间较长，速度较慢，但磁带能存储的信息容量大，价格便宜，便于携带，互换性好，是大、中计算机系统中常用的外存储器。

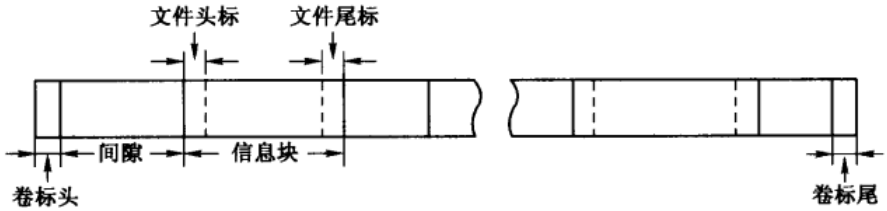


图 2-4 磁带存放信息的格式示意

磁带机的主要技术参数如下。

- 带速：在磁带机的初始加速之后，磁带以稳定的速度运动，磁头可进行读写操作。该速度被称为磁带机的带速，如高速磁带机的带速为 4～5 m/s。
- 记录密度：磁带每寸所能记录的字节数，称为磁带机的记录密度，单位是 Bpi，如磁盘机的记录密度分为 800 Bpi、1600 Bpi、3200 Bpi、6250 Bpi 等几种。

- 数据传输率：磁带机在单位时间内所能传送信息的数量。它是记录密度与带速之积，如某磁带机的记录密度为 1600 Bpi，带速为 18.75 in/s，则数据传输速率为 1600×18.75＝30 000 Bps。

磁带有各种不同的规格，每种规格的磁带宽度、长度、记录密度及宽度方向的磁道数各不相同，常用磁带的宽度为 1/2 in，含 9 个磁道，每个磁道上有一个磁头，记录密度为 1600 Bpi。

3．磁盘存储器

磁盘存储器按其载磁体的基片是“硬”的（铝合金圆盘）还是“软”的（塑料圆盘），分为硬磁盘存储器和软磁盘存储器两种。简称硬盘机和软盘机。这两种磁盘机组成及工作原理大致相同，故下面先介绍这两种磁盘机的共性问题，然后分别介绍它们的不同之处。

（1）磁盘机的结构

磁盘机的结构原理图如图 2-5 所示。它由磁盘驱动器、磁盘机接口板及磁盘等所组成。

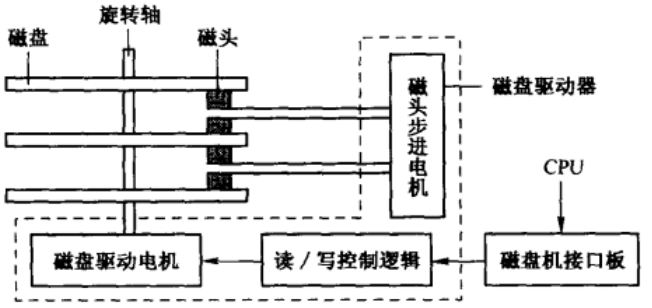


图 2-5　磁盘机的结构原理图

在如图 2-5 所示中，磁盘是存储信息的载体。每片磁盘可有两个盘面。每个盘面上有一个读写磁头，用于读取或写入该盘面上的信息。磁盘驱动器是实现读写操作的设备。它包含磁头步进电机、磁盘驱动电机及控制读写的逻辑电路等。磁盘机接口板是连接 CPU 与磁盘驱动器的部件，接收来自 CPU 的控制命令，发出使磁盘驱动器进行操作的控制信号。

磁盘表面的信息存放格式如图 2-6 所示。每个盘面（或称记录面）上有几十条到几百条同心圆磁道，由外向里分别为 0 磁道、1 磁道……N 磁道。每条磁道上又分成若干扇区。每个扇区存放若干字节的信息。磁盘上的信息以块作为存取单位。一个信息块可以是一个扇区或是多个扇区。磁盘机与内存交换信息时，应给出磁盘的盘面号、磁道号、扇区号及存取信息块的长度。这些参数实际上就是访问磁盘存储器的“地址”。这样，当主机访问磁盘存储器时，先根据给定的盘面号，启动该盘上的读写磁头处于读写状态；然后由磁头步进电机将上述磁头移动到给定磁道号的磁道上，磁盘在驱动电机的驱动下旋转，当给定扇区号的扇区进入上述磁头之下时，便可以从该磁头中存取信息，直到将给定长

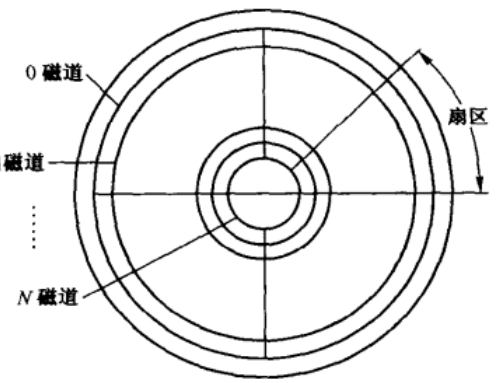


图 2-6　磁盘表面的信息存放格式

度的信息块全部存取完毕。

(2) 磁盘机的主要技术指标

磁盘机的主要技术指标有 4 项。它们是记录密度、存储容量、寻址时间及数据传输速率。

记录密度又称存储密度，一般用磁道密度和位密度来表示。磁道密度是指沿磁盘半径方向，单位长度内磁道的条数，单位是 TPI（道每英寸）。位密度是指沿磁道方向，单位长度内存储二进制信息的个数，单位是 BPI（字节每英寸）。由于各个磁道上的存储容量是相同的，且越靠内侧的磁道越短，故内侧磁道上的位密度要比外侧磁道上的密度高。有时也用磁道密度与平均密度来描述磁盘机的记录密度。

磁盘机的存储容量是指它所能够存储的有用信息的总量，单位是字节。存储容量可按下列公式计算，即

$$C = n \times k \times s \times b$$

式中，n 为存储信息的盘面数，k 为盘面上的磁道数，s 为每一磁道上的扇区数，b 为每个扇区可存储的字节数。

寻址时间是指磁头从启动位置到达所要求的读写位置所经历的全部时间。它由寻道时间 t_s（又称查找时间）和平均等待时间 t_w 两部分组成。寻道时间是指磁头找到目的磁道所需要的时间。平均等待时间是指所读写的扇区旋转到磁头下方所用的平均时间。因为磁头等待不同的扇区所用的时间不同，故一般取磁盘旋转一周所用时间的一半作为平均等待时间。显然，平均等待时间与磁盘的转速有关。寻道时间由磁盘机的性能决定，由磁盘生产厂家给出。

数据传输速率是指磁头找到地址后，每秒读出或写入的字节数。其计算方法是：一个磁道上存储的字节数除以磁盘转一周所需的时间。

(3) 硬盘与软盘

硬盘是以铝合金圆盘为基片，且上下两面均涂有磁性材料而制成的磁盘。它质地坚硬，可将多个盘片固定在一根轴上以组成一个盘组。一台硬磁盘机可有一个或多个盘组。硬盘上的读写磁头大多数是浮动的，可沿着盘面的径向移动。目前，常用的硬盘是温彻斯特盘（简称温盘）。其直径有 5.25 in 和 3.5 in 两种类型。计算机采用直径为 3.5 in 和 2.5 in 的小型温盘，并将硬盘片、磁头、电机及驱动部件全做在一个密封的盒子中，因而具有体积小、重量轻、防尘性好、可靠性高及使用环境比较随便等特点。与软盘相比，硬盘具有存储容量大、存取速度快（盘片转速可达 10 000 r/min）等优点。但硬盘多固定于主机箱内，故不便于携带，价格也高于软盘。

软盘（Floppy Disk）是以塑料圆盘为基片，上下两面均涂有磁性材料而制成的磁盘。它质地较软，需封装在一个专用套内。其中，驱动旋转孔用于固定盘片，使盘片可以在驱动电机的带动下旋转，以便磁头进行读写信息。磁头读写孔是磁头与软盘的接触区域。在该区域内磁头可前后移动，以便对每一个磁道的开始和结束位置进行定位。写保护槽口用于对软盘中的信息进行保护，如在 3.5 in 的软盘中，若将写保护槽口封死，则该软盘只能读出信息，而不能写入信息或对软盘中的内容进行修改。标签是生产厂家的标志及软盘的规格和型号等。

常用的软盘直径为 3.5 in。与硬盘相比，软盘的存储容量较小、存取速度较慢（如 3.5 in

软盘的转速约为 300 r/min），但软盘携带方便、价格低廉、便于保存，是目前使用最广泛的辅助存储器。

2.2.4　光介质存储器

光盘是指利用光学原理进行读写信息的圆盘。目前广泛使用的有视频光盘，用于存储视频信号；还有激光唱片，用于存放数字音频信号。计算机所用的光盘用于存储数字信号。光盘存储器（Optic Disk Storage）是利用激光束在光盘表面上存储信息，并根据激光束反射光的强弱来读出信息的。

常用的光盘存储器可分为下列三种类型。

（1）只读型光盘存储器

只读型光盘存储器（Compact Disk-Read Only Memory，CD-ROM）的盘片由生产厂家预先写入程序或数据，出厂后用户只能读取而不能写入或修改。

（2）只写一次型光盘存储器

只写一次型光盘存储器（Write Once and Read Memory，WORM）的盘片可由用户写入信息，但只能写入一次。写入后，信息将永久性地保存在光盘上，可以多次读出，但不能修改。

以上两种光盘存储器就其功能而言，类似于半导体存储器中的掩膜 ROM 和 PROM，但实现原理却完全不同。它们实现读写的基本原理是：写入信息时，将激光聚焦成直径不超过 1 μm 的激光束，照射到记录介质上，使其局部加热到能把介质熔化，形成一个小凹坑，改变了光学特性；读出信息时，光电检查电路根据被激光照过的介质和没有被照过的介质对光的反射率不同，便可读出所存储的信息。

（3）可重写型光盘存储器

这种光盘存储器类似于磁盘，可以重复读写。其写入和读出信息的原理随使用的介质材料的不同而不同。用磁光材料记录信息的原理是利用激光束的热作用改变介质上局部磁场的方向来记录信息，再利用磁光效应来读出信息。

光盘存储器具有下列突出的优点：

- 存储容量非常大，如一片 5.25 in 的光盘可以存储 1 GB 的信息；
- 可靠性高，如不可重写光盘（CD-ROM，WORM）上的信息几乎不可能被丢失；
- 存取速度高。

光盘存储器的这些优点已受到人们的高度重视，并已应用于计算机系统中。

2.3　输入输出设备及接口

输入输出设备是实现计算机系统与人（或其他系统）之间进行信息交换的设备。通过输入设备可以把程序、数据、图像甚至语音送入计算机中，通过输出设备可以把计算机处理的结果显示或打印出来呈现给用户。输入输出设备通过其接口（Interface）实现与主机交换信息，完成输入或输出操作。

2.3.1 输入设备

1. 键盘

键盘（Key Board）是最常用的输入设备。它由一组开关矩阵组成，包括数字键、字母键、符号键、功能键及控制键等。每个按键在计算机中有它唯一的代码。当按下某个键时，键盘接口将该键的二进制代码送给计算机的主机，并将按键字符显示在显示器上。当计算机操作人员击键速度过快，CPU 来不及处理时，CPU 先将键入的内容送往主存储器的键盘缓冲区，然后再从该缓冲区中取出进行分析和执行。当前，键盘接口多采用单片微处理器，由它控制整个键盘的工作，如上电时对键盘的自检、键盘扫描、按键代码的产生、发送与主机的通信等。

在计算机中，常见的键盘有两种：一种是 101 键盘；另一种是 104 键盘。104 键盘是在 101 键盘上增加了 3 个可直接操作 Windows 9x 菜单的按键。其中，两个键分别是位于空格键两侧的 Alt 键与 Ctrl 键之间的“Windows”键；另一个键是位于右边“Windows”键与 Ctrl 键之间的“Application”键。

2. 鼠标

鼠标（Mouse）是一种手持式屏幕坐标定位设备，常用在下拉式菜单中选择操作项或计算机辅助设计系统中的做图。常用的鼠标有两种：一种是机械式的；另一种是光电式的。

机械式鼠标的底座上装有一个可以滚动的金属球。当鼠标器在桌面上移动时，金属球与桌面因摩擦而发生转动。金属球与 4 个方向的电位器接触，可测量出上、下、左、右 4 个方向的相对位移量，用以控制屏幕上光标的移动。光标和鼠标的移动方向一致，而且移动的距离也成比例。

光电式鼠标的底部装有两个平行放置的小光源（小灯泡）。当它在桌面上移动时，光源发出的光经桌面反射后被鼠标接收为移动信号。该移位信号送入计算机，使屏幕上的光标随之移动。光电式鼠标的其他方面均和机械式鼠标一样。

按鼠标的按钮数目可将鼠标分为两键鼠标和三键鼠标。

两键鼠标又叫 MS Mouse，由微软公司设计提倡的鼠标，只要左、右两个按键即可。该鼠标可以说是默认的鼠标标准。

三键鼠标又叫 PC Mouse，是由 IBM 公司设计提倡的鼠标，在原有的左、右两键当中增加了第三个按键，即“中键”。很多软件也经常使用到中键，特别是绘图软件、三维射击游戏及上网浏览时，鼠标“中键”确实使操作事半功倍。

3. 其他输入设备

计算机的输入设备按功能可分为下列几类。

- 字符输入设备：键盘；
- 光学阅读设备：光学标记阅读机、光学字符阅读机；
- 图形输入设备：鼠标、操纵杆、光笔；
- 图像输入设备：摄像机、扫描仪、传真机；
- 模拟输入设备：语音、模数转换。

以上对属于字符和图形输入设备的键盘和鼠标做了介绍。下面将简要说明其他三类输入设备中的某些设备的功能和基本工作原理。

光学标记阅读机是一种利用光电原理读取纸上标记的输入设备，如用作计算机评卷记分的输入设备、广泛用于商品和图书管理的条形码读入器等。条形码读入器的基本原理是：用粗细不同的明暗条纹表示号码，当条形码读入器顶端的光源扫描明暗条纹时，便产生了长短不同的电压波形，经译码后就是所读条纹的编码。因此，只要把数字、符号和字母变成条形码，就可以很方便地将它读入到计算机中。

图形（图像）扫描仪可以直接将图形（图像）输入到计算机中，目前国内已开始把图像输入用于一些图像资料库管理中，如人事档案中的照片输入。

现在，人们正在研究如何使计算机具有人的“听觉”和“视觉”，即让计算机能听懂人的说话，看懂人所写的文字，从而能以人们接收信息的方式接收信息。为了达到这个目的，一系列的相应学科正在形成与发展，其中包括模式识别、人工智能、信号及图像处理等，并在这些学科的基础上产生语音识别、文字识别、自然语言理解与机器视觉等学科。语音和文字输入技术的实质是使计算机从语音的声波及文字的形状中领会所听到的声音或见到的文字的含义，即对声波与文字的识别。

2.3.2　输出设备

常用的计算机输出设备有以下几种。

- 打印机：分为击打式和非击打式两类；
- 显示器：分为字符显示器、图形显示器、液晶显示器及等离子显示器；
- 绘图仪：分为平板式和滚筒式两类。

下面简要说明这几种输出设备的基本工作原理及其相关知识。

1. 打印机

打印机（Printer）是计算机最基本的输出设备之一。它能将计算机的处理结果打印在纸上。打印机按印字方式可分为击打式和非击打式两类。击打式打印机（Impact printer）是利用机械动作打印“字体”，使打印钢针与色带和打印纸相撞，在纸上印出字符或图形。根据“字体”的结构。击打式打印机又分为活字式打印和点阵式打印。活字式打印是把每一个字刻在印字机上，可以是球形、菊花瓣形、鼓轮形等各种形状。点阵式打印是利用由打印钢针组成的点阵来表示要打印的字符，每个字符可由 m 行×n 列的点阵组成。

点阵式打印机被简称为针式打印机。其打印原理是：打印头上有一列钢针，打印机在打印驱动程序的控制下，使每个钢针在适当的位置上动作，打印出码点。例如，一个字符由7×8点阵组成，则打印头上有一列 7 个钢针，每打印一个字符，则需打印头要打印 8 次。显然，点阵的码点越多，打出来的字越漂亮，一般汉字是由 24×24 点阵组成的。针式打印机常以打印头上的钢针数目命名，如 9 针打印机和 24 针打印机。针式打印机具有结构简单，价格低、打印内容不受限制（可打印字符、汉字及各种图形）等优点，是目前计算机上最常用的打印机。

非击打式打印机（Nonimpact Printer）用各种物理或化学的方法印刷字符，如静电感应、电灼、热敏效应、激光扫描及喷墨等。其中，常用的是激光打印机（Laser Printer）和喷墨式打印机（Inkject Printer）。它们都是以点阵的形式组成字符和各种图形的。激光打印机接收来自 CPU 的信息，然后进行激光扫描，将要输出的信息在磁鼓上形成静电潜像，并转换成磁信号，使碳粉吸附到纸上，经加热定影后输出。喷墨式打印机是靠墨水通过精制的

喷头喷到纸面上形成字符图形的。非击打式打印机具有打印速度高、印字质量高及运行无噪声等优点。但其价格较针式打印机高。

2. 显示器

显示器（Display）是计算机必备的输出设备，常用的有阴极射线管显示器、液晶显示器及等离子显示器。液晶显示器和等离子显示器是平板式的，体积小、功耗少，主要用于笔记本计算机中。

阴极射线管显示器可分为字符显示器和图形显示器。字符显示器只能显示字符，不能显示图形，一般只有两种颜色。图形显示器不但可以显示字符，而且也可以显示图形和图像。图形是指工程图，即由点、线、面、体组成的图形；图像是指景物图。它们都是由显示器上的像素（光点）所组成的。无论是哪种显示器，其工作原理都与电视机相似，即由阴极电子枪发射电子束，电子束从左向右、从上而下地逐行扫描荧光屏。每扫描一遍屏幕，便称为刷新一次，只要两次刷新的时间间隔少于 0.01 s，人们在屏幕上看到的就是一个稳定的画面。

显示器通过“显示器接口（常称显示控制卡）”及总线与主机连接，要显示的信息（字符或图形）由显示缓冲存储器送入显示器接口，通过控制电子束的强弱，在屏幕上形成不同亮度的光点（灰度）；通过控制电子束发射光点的位置，在屏幕上形成不同的点阵，从而显示出明暗层次的画面。因此，显示器的两个重要技术指标是：一屏有多少个光点（像素）被称为显示器的分辨率；光点亮度的深浅变化层次，被称为灰度，它可以用颜色来表示。分辨率和灰度的级别决定了所显示图像的质量。

常用的显示控制卡有多种，如 CGA 卡、EGA 卡、VGA 卡、MGA 卡等。现以 VGA（Video Graphics Array)视频图形显示控制卡为例，说明它的分辨率和灰度。标准的 VGA 显示卡的分辨率是 640×480，灰度是 16 种颜色；增强型的 VGA 显示控制卡的分辨率为800×600、960×720 个像素点，灰度可为 256 种颜色。所有显示控制卡只有配上相应的显示器和显示软件，才能发挥它们的最大效能。

3. 绘图仪及其他输出设备

绘图仪是输出图形的重要设备，在计算机辅助设计（CAD）系统中获得广泛的应用。常用的绘图仪有两种：一是平板式绘图仪；另一是滚桶式绘图仪。

平板式绘图仪将绘图纸固定在平板上，计算机执行绘图程序时，需将加工成的绘图信息送入绘图仪。绘图仪产生驱动 x 方向和 y 方向的步进电机的脉冲，使绘图笔在 xy 平面上运动，并控制绘图笔的起落，从而在图纸上绘出图形。滚桶式绘图仪是将图纸卷在一个滚桶上的，并在计算机的控制下，图纸沿垂直方向随滚桶卷动，绘图笔则沿水平方向移动，图纸卷动一行，绘图笔便绘制一行。与平板式绘图仪相比较，滚桶式绘图仪具有结构紧凑、占地面积小、重量轻、绘图速度快等优点，但对绘图纸要求较高。

2.3.3 输入输出设备接口

以上介绍的外存储器、输入设备和输出设备被统称为计算机的外部设备，简称外设。主机与外设之间是通过“接口”交换信息的，每一台外部设备都有各自的“接口”或称适配器（Adapter）、设备控制卡（Device Control Card）或输入输出控制器，如图 2-7 所示。

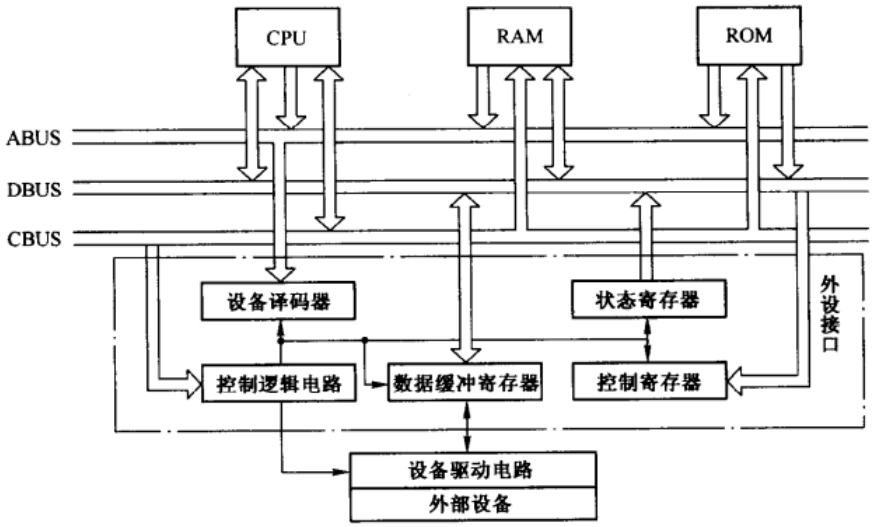


图 2-7　主机与外设的连接

尽管不同外设接口的组成及任务各不相同，但它们要实现的基本功能大致相同。一般来说，任何外设接口都必须具有下列基本功能。

(1) 实现数据缓冲。在外设接口中设置若干个数据缓冲寄存器，在主机与外设交换数据时，先将数据暂存在该缓冲器中，然后输出到外部设备或输入到主机。

(2) 能够将外设的工作状态“记录”下来，并“通知”主机，为主机管理外设提供必要的信息。外设工作状态一般可分为“空闭”、“忙”和“结束”三种。这三种状态在接口中用状态寄存器记录下来。

(3) 能够接收主机发来的各种控制信号，以实现对外设的控制操作。为此，在接口设置了控制寄存器，以存放主机发来的控制字。

(4) 能够判别主机是否选中该接口及其所连接的外部设备。众所周知，一个计算机系统往往配置有多台外部设备，如 CRT 显示器、针式打印机及磁盘机等。这些外部设备都有各自的设备号，主机根据这些设备号来确定与哪一台外设交换数据。为识别该设备号（或端口地址），接口中设置了设备译码器。

(5) 实现主机与外设之间的通信控制，包括同步控制、中断控制等。为此，在接口中包含有控制逻辑电路。

下面以针式打印机为例说明接口的工作过程。当主机需要用打印机输出数据时，打印机接口的工作过程如下。

(1) 主机通过地址总线向接口发送设备号，经设备译码器译码，选中该打印机接口；

(2) 主机测试打印机接口中状态寄存器的状态，判别打印机所处的工作状态：若测得打印机处于“忙”状态，则表明打印机正忙于打印输出，因而不能接收新的打印数据，直到打印机由“忙”状态转换到“结束”状态；若测得打印机处于“结束”状态，则表明前一次打印已结束，可以接收新的打印数据；若测得打印机处于“空闲”状态，则表明打印机尚未启动，需先启动打印机，然后才能接收新的打印数据。

(3) 当确定打印机可以接收新的打印数据后，主机通过数据总线向接口的数据缓冲寄存

器送入要打印的数据。

(4) 主机向接口中的控制寄存器送入控制字，通过控制逻辑电路发出打印输出所需要的控制信号。在该信号的控制下，驱动打印机钢针把数据缓冲寄存器中的内容打印在纸上。在打印过程中，打印机进入“忙”状态，直到打印完毕后才转入“结束”状态。

以上工作过程都是在一定的输入输出控制方式下，通过程序的执行完成的。

接口的分类方法很多。按信息传送格式，接口可分为串行接口和并行接口。串行接口以串行方式（即逐位传送方式）与外设交换数据。并行接口则以并行方式（即按字节或字传送方式）与外设交换数据。按使用方式，接口可分为可编程接口与不可编程接口，还有专用接口与通用接口之分。

小结

硬件系统是计算机系统的物质基础，一般指用电子器件和机电装置组成的计算机实体，是计算机系统的执行部分。

本章主要讲解了 CPU、存储器和输入输出设备等方面的知识。

在 CPU 方面，具体讲解了 CPU 是由运算器、控制器两部分构成的，主要应掌握运算器的基本构成和工作过程、控制器的基本构成和工作过程。

在存储器方面，讲解了存储器的基本概念、分类、性能指标，主要应掌握半导体存储器、磁介质存储器和光介质存储器的工作原理及各自优点。

在输入输出设备方面，主要讲述的是常用的输入设备键盘和鼠标器的工作原理、常用输出设备显示器和打印机的工作原理。

习题

1. 什么是 CPU？它由哪几部分组成？
2. 控制器有几个部件组成？各部件的功能是什么？
3. 什么是计算机的指令？它由几部分组成？
4. 什么是计算机的指令系统？一般包括多少种功能的基本指令？
5. 什么是存储器？它有什么特点？
6. 什么是 ROM？什么是 RAM？
7. 目前主要有几种 ROM？它们各自有什么特点？
8. 写出 GB、MB、KB 与 B 的换算关系。
9. 什么叫磁盘的磁道？什么叫磁盘的扇区？
10. 磁盘存储器主要技术指标是什么？
11. 已知磁盘机的盘组由 6 块盘片组成，有 11 个盘面可记录数据（一般最上一块盘片的上面或最下一块盘片的下面不记录数据），每面分 512 个磁道，每道分成 32 个扇区，每个扇区存储 512 字节信息，问该磁盘机的存储容量为多大（以字节为单位）？
12. 为什么光盘也称作 CD 盘？CD-ROM 表示什么含义？
13. 输入设备按功能可分为几类？常用的输入设备有哪些？
14. 什么是 CRT 显示器的分辨率和灰度？

第 3 章　计算机系统的软件

本章要点

- ☑ 计算机软件系统：系统软件、应用软件
- ☑ 程序设计语言：机器语言、汇编语言、高级语言
- ☑ 操作系统
- ☑ 数据库系统

3.1　计算机软件概述

计算机系统由硬件系统和软件系统两大部分组成。通常，把不装备任何软件的计算机称为裸机。而只有装配上软件的计算机才是完整的计算机系统。什么是软件？软件有什么功能？软件包括哪些内容？

3.1.1　软件的概念

人们针对某一需要而为计算机编制的指令序列称为程序。程序连同有关的说明资料（如程序清单、说明书、使用及维护手册等）统称为软件。

每一台计算机都设置了完成一组基本操作的电子线路，即任何一种计算机都有它自己的指令系统，计算机可以直接执行的也仅仅是它的指令系统中的指令。因此，从这个意义上而言，任何一种计算机系统的硬件部分的功能都是固定的、十分有限的。为了使计算机能连续、自动地进行工作，为了使用户能方便地应用计算机来完成各种不同的任务，就必须为它编写相应的程序。

计算机软件的内容非常丰富，对其严格分类比较困难。一般从计算机系统角度划分，主要有系统软件和应用软件两大类。

3.1.2　系统软件

系统软件是指管理、控制和维护计算机及其外部设备，提供用户与计算机之间界面等方面的软件，是用来扩大计算机的功能、提高计算机的工作效率、方便用户使用的软件。具有代表性的系统软件有操作系统、故障诊断系统、数据库管理系统及各种程序设计语言的编译系统等。这些软件大多由计算机研制单位或生产厂家向用户提供。

系统软件也是计算机系统的一部分“设备”。当然，这部分“设备”的一部分与计算机的硬件设备不同。它们不是电子线路构成的电子装置，而是存储在机器内的程序。故可称这部分“设备”为“软设备”。

1. 操作系统（Operating System）

操作系统是一个功能强、规模大的管理程序，是最基本的系统软件。它在计算机系统中占有重要的地位。操作系统对计算机的硬件设备、运行程序等各种资源进行管理，包括对其他系统软件和大量的应用软件的管理，以便尽可能合理地调用硬件和软件资源，使整个计算机系统协调、高效率地运行。当计算机配置了操作系统后，用户不再直接对计算机硬件进行操作，而是利用操作系统所提供的命令和其他方面的服务去操作计算机。因此，操作系统是用户操作和使用计算机的强有力的工具，也是用户与计算机之间的接口。微型机常用的操作系统有 DOS、Windows、Linux 等，中、大型机多用 UNIX 操作系统。

2. 故障诊断系统（Fault Diagnostor）

为了判断计算机是否正常工作，或者如果有故障，则希望能判明出错的部位，故许多系统配置了故障诊断程序。在日常维护计算机时，维护人员则可通过故障诊断程序对计算机进行测试或帮助寻找故障。

3. 数据库管理系统（Data Base Management System）

数据处理是当前计算机应用的一个重要领域。计算机的效率主要是指数据处理的效率。有组织地、动态地存储大量的数据信息，而且又要使用户能方便、高效地使用这些数据信息，是数据库管理系统的主要功能。

目前，在国内应用较多的数据库管理系统有 FoxPro、SQL Server 2000、Oracle 等。

4. 编译系统

计算机在执行程序时，首先要将存储在存储器中的程序指令逐条地取出来，经过译码后向计算机的各部件发出控制信号，使其执行规定的操作。计算机的控制装置能够直接识别的指令是用机器语言编写的，而用机器语言编写一个程序是件相当繁重而艰难的工作。实际上，目前几乎很少直接用机器语言编写程序。一般用户现在都使用某种程序设计语言（即高级语言）来编写程序，但是用这些高级语言编写的程序 CPU 是不认识的，必须经过翻译变成机器指令后才能让计算机执行。而负责这种翻译的程序被称为编译程序，也称为编译系统。目前，常用的高级语言有 FORTRAN 语言、C 语言、Visal Baisc 语言、Delphi 语言、Java语言等。不同的高级语言有各自的编译系统。另外，高级语言是不依赖于具体机器的通用语言。但是，不同的计算机，由于它们的机器结构和机器语言不同，因此对于同一种高级语言，不同计算机配置的编译程序是不同的。

以上仅介绍了几种常见的系统软件，目前系统软件正极为迅速地发展和丰富，从而使计算机的功能越来越强。

由于给计算机配置了各种系统软件，故不仅使机器的功能得到了很大的扩充，而且给用户使用计算机提供了很大的方便。因为用户已不再需要像过去那样直接与机器打交道，而只是通过操作系统来使用计算机，故从使用的观点上看，用户现在所见到的，已不再是一台实在的机器（Bare Machine），而是一台具有更强功能、使用方便灵活的“指令系统（即各种命令）”的虚拟计算机（Virtual Computer）。其组成如图 3-1 所示。

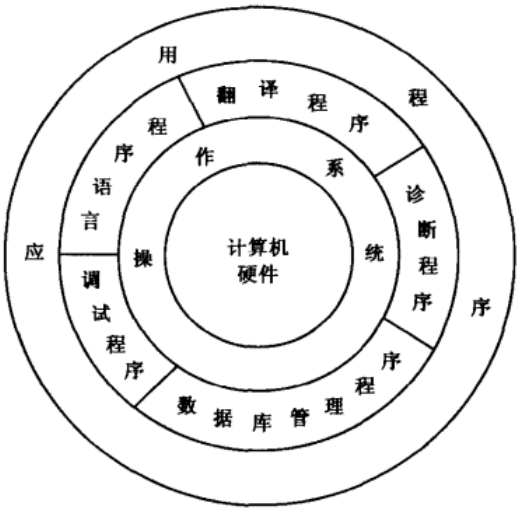


图 3-1　虚拟计算机组成

3.1.3　应用软件

应用软件是指用户专门为解决某个应用领域内的具体问题而编制的软件（或实用程序）。计算机的应用几乎已渗透到了各个领域，所以应用程序也是多种多样的。目前，应用软件常用的有以下几大类：

- 专用计算程序，即各种专用的科学计算程序、工程结构计算和分析程序。这些都是设计研制计算机最初要解决的问题。
- 文字处理软件，即用于输入、存储、修改、编辑、打印文字材料（文件、稿件等）等软件，如 Word、WPS 等。
- 信息管理软件，即用于输入、存储、修改、检索各种信息等软件，如工资、人事管理软件、图书资料自动检索系统、计划管理软件等。这种软件发展到一定水平后，各个单项的软件相互联系起来，使计算机和管理人员组成一个和谐的整体，各种信息在其中合理地流动，即可形成一个完整、高效的管理信息系统，简称 MIS。
- 辅助设计软件，即用于高效地绘制、修改工程图纸，进行设计中的常规计算，帮助设计者寻求较好的设计方案等的软件，其中最著名的如 AutoCAD。
- 实时控制软件，即用于随时收集生产装置、飞行器等的运行状态信息，以此为依据按预定的方案实施自动或半自动控制，可安全、准确地完成任务。

随着应用计算机领域的不断扩大，技术水平的不断提高，应用程序也越来越丰富。许多行业和部门都逐步积累了一大批不同用途和功能的应用软件。为了使用户避免重复劳动，提高工作效率，目前人们正力图使各种应用程序标准化、模块化，并将解决某一类问题的各种程序组合成应用软件包，作为软件产品供用户选用。

当前，各种计算机软件的数量很大，由于受存储容量的限制，不可能在一台计算机中装入过多的软件，因而在一台计算机出厂时，厂家仅在其中配置一些较广泛使用的基本系统软件和少量的应用软件，而用户可视其工作的需要再逐步添置其他软件。

3.2 程序设计语言

编写计算机程序所用的语言即程序设计语言，是人与计算机之间交换信息的工具，是软件系统的重要组成部分。它一般分为机器语言、汇编语言和高级语言三类。

3.2.1 机器语言

机器语言是计算机的第一代语言。它全部由 0、1 代码组成，是能够直接被机器所接受的语言，也称为手编语言，通常随计算机型号的不同而不同。机器语言中的每一条语句（即机器指令）实际上是一条二进制形式的指令代码，由操作码和地址码组成。

机器语言不容易记忆，程序编写难度大，调试修改繁琐，但执行速度最快，它是一种面向机器的程序设计语言。

[例 3-1] 下面是一个可运行在 Intel-8086 微处理器中的程序。它所完成的功能是把两个已分别存放在内存地址为 51H 和 52H 单元中的两个数相加，然后把两数相加后的结果存放到 50H 单元中去。

```
10110000
01010001
10110011
01010010
00000010
11000011
10100010
01010011
00000000
```

可以很容易看出，这个用二进制代码编制的程序存在难写、难读、难修改这几个不仅难以克服，而且难以容忍的缺陷。由于辨认冗长的二进制代码消耗了人们大部分的精力和时间，很难使人们能在程序编制的方法上下工夫，所以极大地影响了程序的质量和应用范围。因此，在实际使用中很少有人直接用二进制代码编写程序。为了改善二进制代码冗长难辨的不足，人们往往都使用十六进制代码或八进制代码来表示机器码。例如，前面的二进制代码程序可以用以下十六进制代码程序表示，即：

```
B0
51
B3
52
02
C3
A2
53
00
```

显然，这个程序较前面的程序短了许多（长度仅为前面程序的 1/4），因此较容易阅读和编写，也易于检查。

然而，计算机只能识别二进制代码，所以用十六进制代码程序直接控制计算机是不可能的，而必须翻译为二进制代码。这种转换工作对于任何人来说无疑都是极其繁琐的，而且很容易出错。为解决这些问题，故应先编制出一个能将十六进制数转换为二进制数的“十六进制代码装入程序”，然后将其翻译为二进制代码后装到计算机中，并且使它常驻在存储器中，成为计算机系统的一部分。每当计算机需要输入用户编制的程序或用户的数据时，计算机便自动地运行此程序，将所有的输入数据转换为二进制代码并存于计算机内存中，从而使得用户可以直接将十六进制代码作为计算机的输入数据。事实上，在多数计算机上都采用这个方法来解决机器码输入困难的问题。这个“十六进制代码装入程序”实际已是一个“翻译程序”了。

尽管如此，程序员仍然感到机器语言程序十分难编、难读、难理解。B0、B3 代表的是什么意思呢？除了死记硬背，似乎没有更好的办法了。为了摆脱机器语言的这个无法克服的缺陷，便出现了汇编语言。

3.2.2　汇编语言

汇编语言是第二代程序设计语言。

在编写程序时，如果我们使用一些可帮助记忆的符号来代替难记、难辨的机器码，则给阅读和修改程序会带来很大的好处。

汇编语言就是用助记符代替操作码，用地址符号代替地址码。它的每一种符号都有一一对应的机器码。正是这种替代，使机器语言“符号化”，所以也称汇编语言是符号语言。

汇编语言与特定类型的机器相对应，也是一种面向机器的语言。事实上，每一个计算机厂家都为自己的机器制定了一套机器码的“助记符”，即汇编语言指令系统。

程序员用汇编语言作为编写程序的工具。用汇编语言编出的程序称为汇编语言程序。如果将例 3-1 中的机器码指令用美国 Intel 公司定义的指令系统中的助记符来表示，则可变为以下形式：

汇编语言程序	机器码程序	含　意
mov al，(51h)	B051	存储单元（51H）内容送入 al
mov bl，(52h)	B352	存储单元（52H）内容送入 bl
add al，bl	02C3	al + bl→al
mov (50h)，al	A25300	al 的内容送入存储单元（50H）

显然，汇编语言与机器语言相比较更容易记忆，程序易读、易检查及易修改，同时，也保持了机器语言编程质量高、执行速度快及占存储量小的优点。但在编制复杂程度较高的程序时，汇编语言还存在着明显的局限性，尤其是这种语言程序依赖于具体的机型，并且通常不同系列的计算机都有自己的一套机器码，所以也就造成了不同系列的机器有不相同的汇编语言的情况，故不具有通用性和可移植性。在某一种机器上调试好的汇编语言程序，换到另一种机器上运行时通常都必须重新编写。它也属于低级语言。

汇编语言的特点之一是它能彻底地反映计算机中 CPU 的内部结构，从而用它可以向计

算机发出该机器可以执行的指令。它能使程序员利用机器的所有功能。但是，另一方面，这就要求程序员熟悉所使用的计算机性能，如必须了解它有哪些寄存器的指令集，确切了解指令是如何对各种寄存器使用的，计算机采用何种寻址方法等。显然，这一切实际上与计算机最终要执行的任务无关，并且意味着程序员在编写程序时必须承担相当大的工作量。

用汇编语言程序控制计算机工作同用十六进制代码控制计算机工作有类似的问题，即计算机无法直接识别，也必须先将其译为二进制代码。这种翻译工作用人工来做显然可行有效，但相当繁琐。这里我们可以借助于前面用计算机来完成将十六进制数转换为二进制数任务的思想来解决这个问题，通过一个称为汇编程序的软件，将其调入计算机内存运行。该程序的输入数据是用户的汇编程序，而输出的便是机器码程序。目前，几乎所有的计算机系统均配有汇编程序。

3.2.3 高级语言

用汇编语言进行程序设计虽然较机器语言方便，但仍然存在两个重要的缺陷：①移植性极差；②要求程序员熟悉所使用的计算机的硬件特性，并且编程工作量通常很大。为解决这两个问题，便出现了高级程序语言（简称为高级语言）。高级语言也被称为第三代语言。

高级语言是20世纪50年代中末期发展起来的面向问题的程序设计语言。高级语言的指令（或语句）一般都采用自然语言，并且使用与自然语言语法相近的自封闭语法体系，这使得程序更容易设计和理解。

高级语言的最大特点是它独立于计算机硬件结构，从而使得同一程序可以在不同的计算机系统上运行，因而它的可移植性较好。高级语言的另一特点是它可以让用户使用面向问题的形式，而不是用面向计算机的形式描述任务，所以使得程序员在编程时无须了解所使用计算机的硬件特性，而可以将大部分精力放在理解和描述所要解决的问题上，并大大地简化了程序的编制和调试，使编制程序的效率得以大幅度提高。

例如，对于例3-1，假设被加数和加数分别已存入在变量A和B中（相当于将这两个数存入某两个存储单元，程序员无须关心具体地址，而由计算机自动分配），并假定将结果存入变量C，则只需要告诉计算机：

$$C=A+B$$

这很像数学中常用的代数式，十分容易理解。它与前面的汇编语言程序相比，既明确又简短，所以用高级语言可以更容易、更快速地编写程序。有人估计，高级语言的编程效率通常要比汇编语言高10倍以上。在程序设计、理解和调试等方面，它也较汇编语言有很明显的优点。

目前，世界上已有数百种高级语言。其中，用得最多的是FORTRAN语言、Pascal语言、BASIC语言、C语言、COBOL语言、PROLOG语言及LISP语言等数十种。

每一种高级语言在执行之前，都需要由机器系统中的编译程序或解释程序翻译成机器所能接受的目标代码，所以高级语言虽然通用性好、容易理解，但是占用内存大、运行速度慢。

取其所长，上述三代语言可分别用在不同的场合。一般科学计算、数据处理采用高级语言比较合适，而实时控制因为速度要求高，则往往可采用汇编语言。

3.2.4　最常见的高级语言及其主要应用领域

1. BASIC（Beginners All-purpose Symbolic Instruction Code，初学者通用符号指令码）

早期的 BASIC（1964 年）属于解释型高级语言，所以程序的编写和调试及学习都十分方便，很适合于初学计算机编程的人员使用，有“启蒙语言”之称。另一方面它也有很大的实用价值，故拥有了众多的用户。随着软件技术的发展，近年来又出现了不少编译型BASIC语言系统，使得它兼有解释型和编译型两类语言的特色。但是，最近，BASIC 语言程序的启蒙地位已被一些新出现的程序设计语言所动摇。在许多高等院校中，由于 BASIC 语言程序结构难以适应结构化程序设计方法的要求，从而使得它已越来越多地被 Pascal 语言所取代；而对于青少年计算机爱好者来说，一种名为 LOGO 的新型程序设计启蒙语言正在迅速普及，且目前有取代 BASIC 语言的趋势。BASIC 语言已很少使用。

2. FORTRAN（Formula Translator，公式翻译程序设计语言）

FORTRAN（1956 年）是一种出现较早的高级语言。编译型高级语言是为解决大量的科学和工程计算问题而产生的。虽然因不符合结构程序设计思想而受到许多计算机专家的强烈批评，但是由于长期以来它已拥有了众多的用户，而 FORTRAN 本身也在改进，所以目前还有很强的生命力。它的规范标准一再翻新。其标准之多为其他任何程序设计语言所不及。由于近年来出现的新型高性能超级计算机多是瞄准大型科学和工程计算而研制的，所以随之而来的配套语言是向量 FORTRAN、并行 FORTRAN 等标准 FORTRAN 的扩充版本，并使其不断地得到新生。

3. COBOL（Common Business-Oriented Language，面向商业的通用语言）

COBOL（1960 年）是一种面向事务处理的高级语言，很适合于编写管理信息系统方面的程序。由于它包括了大量的英文单词，程序的书写接近于英文形式，所以易于理解，在英语国家中十分盛行。但是在我国，由于文种的差异，多数人在程序编写时感觉到不方便，所以它的普及也受到很大的限制。

4. Pascal 语言

Pascal（1971 年）是第一种能系统地体现结构化程序设计思想的程序设计语言，是程序设计语言发展中的一个里程碑。它是为纪念法国数学家布莱斯·巴斯卡（Blaise Pascal）而命名的。这种高级语言是从较老的、用于科学计算的 ALGOL-60 语言衍生而来的，但功能更强且更易于使用。目前，Pascal 语言作为一个能高效率实现的实用语言和一个极好的教学工具语言而已被广泛地采用，且在许多计算机专业课程中都采用它作为教学语言。

5. C 语言

C 语言（1974 年）是一种兼顾“高级”和“低级”语言特点的通用程序设计语言。它具有书写风格独特、表达形式精炼及程序结构清晰紧凑等特点，语言的效率也较高。这里之所以称其为“低级”，是指在设计 C 语言时适当地考虑了机器的硬件结构特点，而不像其他高级语言那样采用独立于机器的思想来设计。因此，它的目标程序质量较高，可进行一些与硬件结构有关的操作，故很适合于编写系统程序。例如，著名的 UNIX 操作系统和 dBASE-Ⅲ数据库管理系统主要用 C 语言编写。

6. PROLOG（PROgramming in LOGic，逻辑程序设计）

PROLOG（1970 年）与一般的高级语言有较大的差别，是所谓的“超高级程序设计语

言”。它主要用于解决一些逻辑判断的问题，从其名称的来历就可以说明这点。用通常的程序设计语言设计程序时，程序必须精确地告诉计算机应该如何去完成任务，而用PROLOG语言去编写程序时，程序员要做的事却是仅告诉计算机一系列事实和规则，而由计算机自己去考虑“如何做”的问题。目前，PROLOG已成为人工智能应用程序设计和开发专家系统的一种重要工具，有许多著名的专家系统都是用它编写的。

3.3 操作系统

3.3.1 操作系统的基础知识

1. 什么是操作系统

简单地说，操作系统是专门用来管理和控制计算机系统的软件和硬件资源，以及方便用户使用计算机、提高计算机系统资源利用率的一组程序。它是用户与计算机之间的接口。用户可通过操作系统使用计算机。

一个计算机系统非常复杂，包括CPU、存储器、各种数据、文件及信息，如何让它们相互协调地工作，如何有效地管理它们，给用户提供方便的操作手段与环境，这些都是操作系统要做的事。

通常把操作系统理解成一个监督系统运行的总管程序。它负责监视系统各种资源的状态和它们的使用情况，决定谁得到这些资源，何时获得，获得多少，并按一定方式调度和分配这些资源，并在资源被使用完毕后，再回收这些资源。

2. 操作系统的功能

操作系统的基本任务有以下两项：首先，操作系统应管理好计算机的全部资源，尽可能地使资源充分地被利用，并有效地进行工作，这里所说的“资源”，既包括CPU、内存、外存和I/O设备等硬件资源，也包括程序、数据等各种软件资源；其次，操作系统担任用户和计算机的接口，给用户提供一个清晰、简捷、易于使用的用户界面，使用户不用过问计算机硬件的具体细节，就能方便地操作使用计算机。

操作系统一般具有CPU管理、任务管理、存储管理、文件管理及设备管理等功能。

3. 操作系统的分类

不同类型的计算机，其操作系统是不同的。操作系统分类有许多方法，按用户数目的多少，可分为单用户和多用户操作系统；按依赖的硬件规模，可分为大型机、中型机、小型机及微型机操作系统；按操作系统的功能，可分为批处理操作系统、分时操作系统及实时操作系统；此外，还有近几年发展起来的网络操作系统和分布式操作系统。

下面就几类常见的操作系统做一简单的介绍。

(1) 单用户操作系统

单用户操作系统一般是为计算机和简单小型机而设计的操作系统。这类计算机规模小，外设简单，计算机的全部资源为一个用户所独占。

(2) 多用户操作系统

对高档微型计算机、小型计算机和大型计算机，由于计算机内、外存容量大，外设较多，计算速度又快，故为了充分发挥计算机资源丰富的优势，提高计算机的使用效率，多数

使用多用户操作系统。多用户操作系统除具有一般操作系统的功能处，还提供多用户分时功能。这意味着一台计算机中可安装多个用户终端，几个人在不同的终端上同时使用一台计算机，使计算机资源为大家所共享。同时，它还提供多任务功能。这就是说，每个用户可以同时进行几项工作，允许许多用户按照自己的要求给每项任务设置不同的优先权。因此，多用户操作系统是一个多用户、多任务及分时的操作系统。

(3) 批处理操作系统

在批处理操作系统之下，用户把一批要计算的问题、数据和作业说明书有序地排在一起形成一个作业流，然后由操作系统来控制。多道批处理操作系统的主要特点是实现计算机工作流程的自动化，且具有计算机资源的利用率高、作业吞吐量大等优点，故在一些计算中心的大、中型计算机系统中，大都配置有这种操作系统，让计算机自动地、顺序地执行，以节省人工操作时间和改善机器的使用情况。

(4) 分时操作系统

分时操作系统是这样的一种操作系统。它使计算机为一组终端用户服务，使得每个用户好像自己有一台自己请求服务的计算机，彼此并不感觉到有别的用户存在。

在分时操作系统中，通常按时间片轮转，即每道程序运行一个时间片。

分时操作系统有以下基本特征。

- 多路性：分时操作系统一般带多个终端，允许若干个用户能基本上同时使用。
- 独立性：用户之间彼此独立操作，互不干扰。
- 及时性：用户的请求能在较短时间内得到响应。
- 交互性：用户要与系统进行人机对话。

与批处理操作系统相比，分时操作系统可较好地解决用户不能直接与计算机“会话”，并及时取得运行结果的弊端，但在资源利用率上，显然前者比后者高。

(5) 实时操作系统

实际生活中存在大量的实时控制问题（如导弹的自动控制、工业生产的过程控制等）和实时信息处理问题（如银行系统、订票系统等）。

实时操作系统为实时控制系统和实时处理系统的统称。所谓实时，就是要求系统及时响应外部事件的请求，在规定的时间内完成对该事件的处理并控制所有实时设备和实时任务协调一致地运行。

分时操作系统通用性强，交互性强，及时性要求一般（通常数量级为秒），而实时操作系统专用性强，交互性弱，及时性要求较高（通常数量级为毫秒或微秒）。

为扩大应用范围，增强处理能力，目前有些操作系统兼有几种处理能力，且近几年来又发展起来一种共享网络资源的操作系统。称这种操作系统为网络操作系统。

(6) 网络操作系统

用通信线路将多台计算机相互连接起来且依据某种网络协议组成的系统被称为计算机网络。网络中的计算机可以是同型的，也可以是异型的；在地域上可以同处一地，也可以分散在相距很远的各个地方。发展计算机网络的目的，在于使网络用户共享计算机网络中的各种资源，充分发挥资源的效益。

为计算机网络配置的操作系统被称为网络操作系统。网络操作系统远比通常单机的操作系统复杂。这是因为：首先，网络中各台计算机都有各自的操作系统，而这些操作系统在种

类和功能上又不尽相同，因此为了在不同计算机或终端之间正确地实现通信，就必须确定一套全网共同遵守的约定（即共同约定信息的格式、信息内容及传输的顺序等事项），称这种约定为通信协议，通信协议由网络软件执行；其次，为方便用户，网络操作系统必须提供多种网络服务，如远程录入服务、分时系统服务及文件传输服务等；最后，除进行全网的资源管理外，网络系统还应有一套确保网络可靠性、安全性的措施。

总之，网络操作系统除了应具备通常操作系统的功能外，还具有许多新的内容。

(7) 分布式操作系统

为分布式处理系统配置的操作系统被称为分布式操作系统。所谓分布式操作系统，是指由若干台无主从关系的处理机组成的系统。系统中的各个处理机虽然是分散的，但它们又是相联的，处理机间可相互进行通信，资源可以共享。在分布式处理系统中，每台处理机可以有自己的局部存储器，能独立完成指派给它的任务。例如，一个程序中的各个子程序可分别指派给各处理机去处理，也可以按功能让各个处理机分级处理一道程序的各个阶段。当然，任务的自动调度与动态分配、各种资源的管理都是在统一的操作系统控制之下进行的。

分布式操作系统的主要特点在于按将其功能加以划分，使每一处理机可完成指定的一部分功能，而所有系统任务可在任一处理机上运行，从而具有高度的并行性并提供了有效的同步方法。

3.3.2 资源管理中的基本方法

操作系统对资源的管理主要有 CPU 管理、内存管理及外设管理等。本节将简单介绍操作系统资源管理中用到的一些基本方法，即 CPU 管理中的多进程方法、内存管理中的虚拟存储技术及外设管理中的中断技术。

1. 进程

(1) 程序的顺序执行和并发执行

早期的计算机系统只能进行单道的运行。也就是说，每一次只能允许一道程序运行。在这道程序运行时，它将独立占用整个计算机系统的资源，计算机按照程序的执行步骤顺序地一步一步执行。在该程序执行完以前，其他程序只能等待。这种程序的执行方式称为程序的顺序执行。显然，在顺序执行过程中，当程序占有 CPU 资源运行时，输入输出等设备资源就闲置；当程序占用输入输出设备资源进行输入输出时，CPU 等资源就闲置。因此，程序的顺序执行方式对计算机系统资源的利用率很低。

程序的并发执行是让多道程序同时运行。但是传统计算机的 CPU 只有一个（现在已有多 CPU 的计算机系统），每一时刻只能执行一条机器指令，故要让计算机并发执行多道程序，就需要把 CPU 资源分成很小的时间片，在每一个时间片内虽然只能执行一道程序运行，但时间片规定的限度一到，就切换到另一道程序运行。这样就可以在单 CPU 上实现多道程序的并发执行。

(2) 进程的概念

进程是 CPU 管理中最基本和最重要的概念。程序是一个静态的代码集合，而进程是程序的动态执行过程。程序在动态运行时需要占用若干资源（如 CPU 资源、内存资源等），进程既是资源分配的基本单位，又是调度和运行的基本单位。进程有三种状态，即就绪、执行和阻塞。进程分配到除 CPU 资源以外所有资源的状态被称为就绪状态。在多道系统中，由

若干个调入的程序（此时被称为进程）构成就绪进程队列。进程分配到包括 CPU 资源在内的所有资源，并且它的程序正在处理机上执行时的状态为执行状态。进程因某种事件（如遇到 I/O 操作请求）而暂时停止运行的状态被称为阻塞状态。

(3) 进程的状态转换

多个并发运行的进程可形成进程的状态转换图。一般来说，进程的状态转换图如图 3-2 所示。初始时，进程处于就绪状态。所有就绪状态的进程组织成一个就绪进程队列。队列是一种按先到先服务方式组织的数据结构。就绪进程队列就是所有处于就绪状态的进程按先到先服务的方式排列。当 CPU 资源空闲时，进程调度程序从就绪进程队列中选出最先到的一个进程，把 CPU 资源分配给该进程，然后该进程转换到执行状态；在 CPU 规定的时间片限度内分配到 CPU 资源的进程运行，当 CPU 规定的时间片限度一到，处于执行状态的进程就被重新放入就绪进程队列，进程调度程序再从就绪进程队列中重新选出最先到的一个进程，把 CPU 资源分配给该进程，然后该进程转换到执行状态。当某个处于执行状态的进程需要等待某事件发生后才能继续运行（如进程需要等待用户输入数据）时，则该进程就处于阻塞状态；当处于阻塞状态的进程所等待的事件发生（如进程需要的数据已得到）时，则处于阻塞状态的进程就重新被加入到就绪进程队列中。

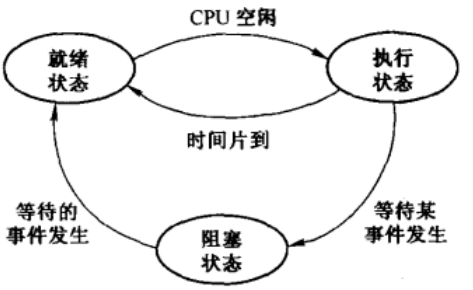


图 3-2　进程的状态转换图

(4) 多进程并发执行对资源利用效率的提高

我们以 3 个并发执行的进程为例说明多进程并发执行对资源利用效率的提高，如图 3-3 所示。假设有进程 A、进程 B 和进程 C 3 个进程。每个进程都由输入数据、处理数据和输出数据 3 个子进程组成。3 个进程的 3 个子进程分别表示为 Ai、Ac、Ao，Bi、Bc、Bo 和 Ci、Cc、Co。在 CPU 的时间片 t_2 时，进程 A 的 Ac 子进程和进程 B 的 Bi 子进程同时运行。由于此时 Ac 子进程只需要 CPU 资源，不需要输入设备资源，而 Bi 子进程只需要输入设备资源，不需要 CPU 资源，所以不会产生矛盾。在 CPU 的时间片 t_3 时，进程 A 的 Ao 子进程、进程 B 的 Bc 子进程和进程 C 的 Ci 子进程同时运行，由于此时 Ao 子进程只需要输出设备资源，Bo 子进程只需要 CPU 资源，Ci 子进程只需要输入设备资源，所以不会产生矛盾。显然，t_3 时刻，计算机系统的输入设备资源、CPU 资源和输出设备资源都同时得到了利用。因此，多道程序的并发执行可以提高计算机系统资源的利用效率。

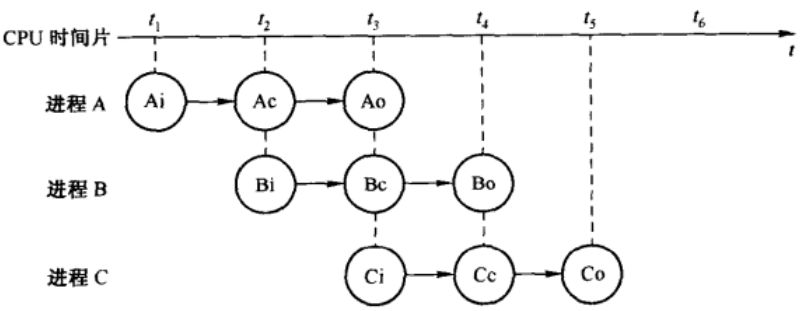


图 3-3　多道程序的并发执行

2. 虚拟存储技术

采用高速缓冲存储器和主存一起构成的层次结构的内存可以大大提高计算机访问内存的速度。目前操作系统的存储管理中广泛采用虚拟存储技术。虚拟存储技术可以为用户提供比实际物理内存大很多的可随机访问的内存空间。

(1) 早期的存储管理技术

早期的操作系统管理内存采用的是单一的连续分区的方法。这时的内存被分为系统区和用户区。系统区用来存放系统软件，如操作系统等。用户区用来存放用户软件。单一连续分区的管理方法是：每当系统要把一个作业调入内存时，首先要检查用户区是否有足够的空间放下要调入的作业。如果有，则内存管理程序调入该作业；如果没有，则内存管理程序不调入该作业，输出相关信息后，从作业队列中重新选择下一个作业。

这样的内存管理方法缺陷很大。其中最主要的问题是，这样的内存管理方法使计算机只能使用实际安装的物理内存。

(2) 虚拟存储

扩大内存的存储空间有两种方法：一种方法是扩大计算机实际安装的内存空间容量，我们把这种内存称为实际内存；另一种方法是借助外存扩大内存的容量，我们把这种方法称为虚拟存储。把实际内存的地址称为实地址或物理地址，虚拟内存的地址称为虚地址或逻辑地址。

虚拟存储的基本思想是：操作系统把外存（如硬盘）和内存上的存储空间分别编号，并建立映射关系。当程序比较大时，操作系统在开始时并不把外存中相对应的全部编号块上的程序和数据调入内存，每当要访问的数据在内存时，就直接访问；每当要访问的数据不在内存时，就首先通过内存和外存之间的映射关系，把相应编号块的数据读入内存，然后再访问该编号块上的数据。这样，通过内存和外存数据的不断交换，虽然在任何时刻可能内存中并没有存储全部用户的程序和全部数据，但并不会影响用户程序的运行。因为虚拟存储是借助外存的存储容量来扩充内存的存储容量的，而外存的存储容量是非常大的，所以虚拟存储方法可以大大扩大内存的存储容量。

(3) 页式存储管理

实地址指的是实际内存中的一个实际地址。在虚拟存储中，虚地址有可能对应的是内存中的一个实际地址，也有可能对应的是外存上的一个实际地址。当把外存中的数据读入内存后，虚地址对应的也是内存中的某个实际地址。因此，虚地址是在程序运行时才能实际确定的内存地址。虚拟存储技术有很多种实现方法，各种方法之间的差别主要是实现虚地址到实地址的转换方法不同。

页式存储管理是实现虚拟存储的一种常见方法。页式存储管理实现虚实地址转换的基本方法是：把存放在外存中的程序和数据按其占用的地址空间分成大小相等的若干个页，把内存的存储空间分成与页的大小相同的若干个块，并建立每个用户程序的页表。用户程序的页表建立了该用户程序的外存页号到内存块号之间的映射关系。当用户程序占用的地址空间比较大时，可能开始时只把外存中的部分页调入内存，在程序运行时，每当要访问的页在内存时，就直接访问；当要访问的页不在内存时，就首先把相应的页调入内存，然后再访问。在外存中连续存放的若干个页，在内存中可以不连续存放。

图 3-4 是页式存储管理基本原理的一个示例。该用户程序由 A_1、A_2、A_3 和 A_4 组成。用

户程序在外存中占用了 4 个页。内存分成大小和页大小相等的若干个块。页表由三项组成，分别是页号、状态和块号。其中，状态 Y 表示对应的页已在内存，状态 N 表示对应的页不在内存。页表建立了用户程序的外存页号到内存块号之间的映射关系。

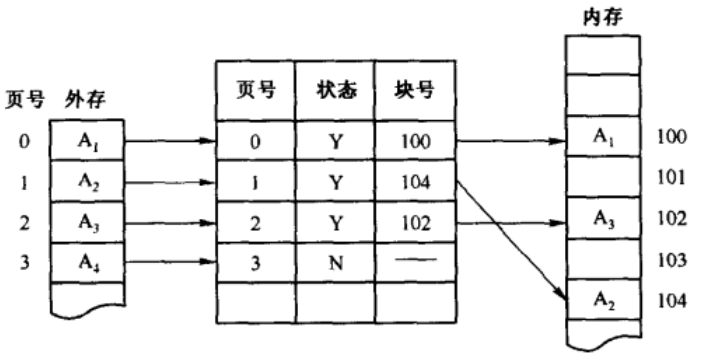


图 3-4　页式存储管理基本原理的一个示例

除页式存储管理外，虚拟存储技术的实现方法还有段式存储管理、段页式存储管理等。

(4) 内存的层次结构

计算机中采用高速缓冲存储器及主存、外存一起构成内存的层次结构。其中，主存是内存层次结构的主体，处于内存层次结构的中间；高速缓冲存储器处于内存层次结构的上部；外存处于内存层次结构的下部。其结论为：

- 高速缓冲存储器与主存构成的层次结构由硬件实现，内存与外存构成的层次结构由操作系统实现；
- 高速缓冲存储器与主存构成的层次结构可以大大提高内存的存取速度；
- 主存与外存构成的层次结构可以大大扩大内存的存储容量。

3. 中断技术

操作系统在进行外部设备管理时可遇到两个非常大的问题：一个问题是外部设备的工作速度远比 CPU 的工作速度低；另一个问题是各个外部设备之间的工作速度快慢不同。有效地解决上述两个问题是操作系统实现外部设备管理的关键。

操作系统进行外部设备管理的基本方法是采用中断技术。中断技术可以有效地解决外部设备管理时遇到的上述两个问题。

(1) 中断的概念

早期的计算机在外部设备工作时，CPU 处于空闲状态。由于外部设备的工作速度比 CPU 的工作速度慢很多，因此在这种工作方式下，宝贵的 CPU 资源白白浪费，整个计算机系统的利用效率很低。

从字面意义看，中断就是暂时停止做某件事情而转去做另一件事情（通常是更紧急的事情），当另一件事情完成后，再转回来完成先前正在做的事情。

在计算机运行程序时，既不能确定什么时间要进行外部设备工作（如输入、输出工作），也不能确定每次外部设备的完成时间。对于这种不能确定发生和结束时间的事件，我们可以借助中断的思想来进行管理。

各个外部设备的工作均是靠相应的外部设备服务程序来指挥的。所谓中断，就是计算机

在运行当前程序过程中，当遇到需要紧急处理的事件时，暂停当前正在运行的程序，转去运行处理紧急事件的程序（通常被称为中断服务程序），当处理紧急事件的程序运行结束后，再自动返回原先正在运行的程序继续运行。

数据的输入、输出是最简单的一种中断。图 3-5 给出了计算机处理输入输出中断的处理流程。把当前正在运行的程序称为主程序。CPU 在运行主程序时，当需要输入数据或输出数据时，操作系统就向相应的输入、输出设备发出一个启动工作信号。此时，输入、输出设备进行启动的准备工作。与此同时，CPU 将继续工作。只是由于主程序需要等待输入数据或输出数据，因此处于阻塞状态，进程调度程序需要重新调度一个新的进程作为 CPU 的主程序。当输入、输出设备准备就绪后，向 CPU 发送一个中断请求信号，CPU 响应中断后，暂停当前正在运行的主程序，转到相应的中断服务程序首地址（称为中断服务程序入口），运行中断服务程序。当中断服务程序运行结束时，CPU 再转回主程序继续运行。

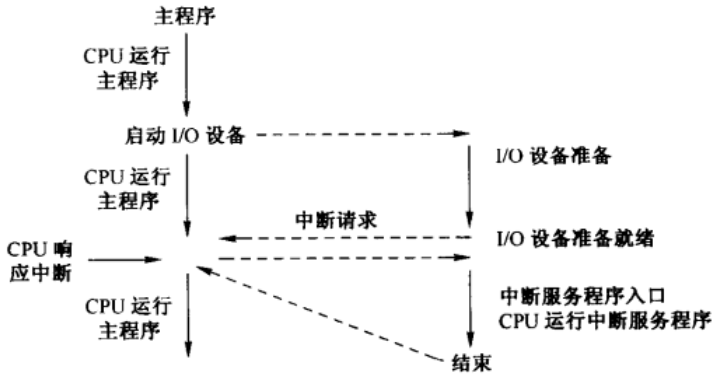


图 3-5 输入输出中断的处理流程

在输入、输出设备进行启动的准备工作（一般这个过程需要数秒）时，CPU 在继续运行主程序（可能是另外一个进程）。因此，CPU 资源没有因输入、输出设备工作而受到影响。

(2) 中断的处理过程

中断处理过程主要包括以下几个步骤。

- 保护中断现场。把被中断进程的 CPU 当时环境（如 PC 的值、通用寄存器的值等）保存到内存的一个特定区间（通常被称为中断保护区）内。
- 识别中断源。根据中断信号带有的特殊标记，判断出当前发生的是哪一种中断。
- 转到相应的中断处理程序。不同的中断处理程序有不同的中断程序入口，根据识别出的中断源，转到相应的中断程序入口。
- 运行中断处理程序。完成相应的中断服务功能。
- 恢复中断现场。恢复保护中断现场时在中断保护区中保存的所有数值，即恢复中断处理过程第一步时保存的所有数值。
- 返回主程序继续运行。

(3) 其他的外部设备管理技术

对外部设备的管理除中断技术外，还有其他的一些技术，即主要是通道技术和外部处理机技术。简单地说，通道技术就是再设计一个独立于 CPU 的功能简单的 CPU，专门处理外

部设备和主机的数据交换事务。因外部设备和主机之间的数据交换就像一个数据通道，所以把此种技术称为通道技术。外部处理机技术再用一台计算机来管理所有外部设备和主机之间的数据交换事务。

3.4　数据库系统

计算机的数据管理技术是随计算机系统的软、硬件的发展而发展的。它经历了自由管理或称人工管理、文件管理及数据库管理等三个阶段。数据库管理实现了数据共享，减少了数据冗余度，支持数据的独立性。通过数据库管理系统可实现对数据的管理与控制。本节将对数据库系统做进一步的介绍，包括数据库系统的特点、数据库管理系统的组成及分类，最后简要介绍几种常用的数据库管理系统。

3.4.1　数据库系统概述

数据库系统（DBS）一般由数据库（DB）、数据库管理系统（DBMS）及用户（包括用户的应用程序）所构成，如图 3-6 所示。用户使用数据库是目的，而数据库管理系统则是帮助用户达到这一目的工具和手段。

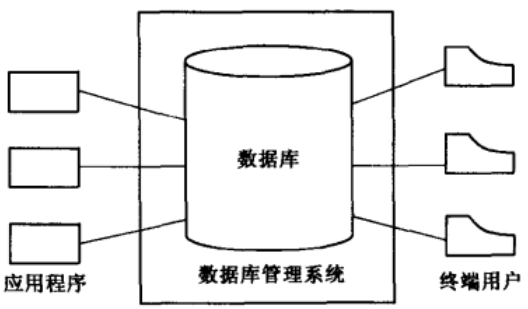


图 3-6　数据库系统结构图

数据库是相关信息或数据的有规则的集合。数据库是一个存储数据的“仓库”。仓库里的数据是按一定的规则存放的，以便于用户对数据进行存/取或修改。如同图书馆中的书库，它是存放各类图书的仓库，这些图书的存放是有一定的规则和结构的，并有相应的藏书模型。若以书卡作为图书馆藏书模型，则可将图书按序、按类地存放在书架上，并使书卡与书架之间建立对应关系（或称映射关系）。读者可凭书卡借阅所需要的图书，而无需关心该图书存放在哪个书架上。图书管理员可根据书卡与书架的映射关系，快速地找到所需要的图书。数据库与此相类似，它需要建立数据模型（如同书卡）及设计数据的物理存储方法（如同图书在书架上的存放方法），并建立数据模型与物理存储之间的映射表。这样，用户（如同读者）就能够通过数据库管理系统（如同图书管理员）找到用户要访问的数据，而不必关心数据具体的存储位置和存储结构，也不必担心由于存储结构的改变而影响数据的访问，即具有数据独立性。这如同改变图书的存放位置却毫不影响读者按书卡借书一样。

由上可知，数据库管理系统是一个数据库的管理软件，其职能是维护数据库，接受并完成用户程序或命令提出的对数据进行输入、编辑、排序、检索、合并及输出等操作的请求。

数据库系统的出现是计算机数据处理技术的重大进步。它有下列主要特点。

- 实现数据共享。数据共享是指多个用户可使用多种语言来存取同一个数据，使数据库中存放的数据可为企业或社会上各部门共同使用。
- 实现数据独立。数据独立是指数据库中存放的数据不依赖于具体的应用程序。程序改变了，数据可保持不变。数据变了，程序可保持不变。数据独立性是数据库系统区别于文件系统的重要特征。
- 减少数据冗余。数据冗余是指数据库中重复的数据。显然，数据库中的重复数据越少，就越能有效地节省存储资源，便于数据一致性的维护。
- 避免数据的不一致性。由于数据库中的数据只有一个物理备份，故数据的访问不会出现不一致的情况。
- 加强了对数据的保护。数据库中加入了安全保密机制，可以防止对数据的非法存取，并可采取一系列措施，实现对被破坏数据库的恢复。

数据库技术是计算机技术中发展最快的重要分支之一。目前，数据库技术的主要研究工作集中在下列几个方面。

- 数据库设计方法。至今，数据库设计方法的工程规范程度仍不高，数据库设计的优劣直接与设计人员的素质相关。如方法复杂，则系统的性能就难以保证。
- 数据库规范理论。规范理论所研究的是数据语义问题和构造规范的数据模型。目前，这方面的工作走向抽象化，已形成一个侧重于数学表达的理论分支。
- 分布式数据库系统。这是数据库技术与计算机网络相结合的产物。其目标在于把数据分散存储于多个网络结点，通过网络通信进行连结。对用户而言，数据的分散性则是透明的（用户无需考虑）。显然，分布式数据库系统有利于不同地点的用户对数据的彼此共享。
- 4GLs及应用生成器。第四代语言（4GLs）源于数据库语言。其主要特点是开发效益高，用户只需指出“做什么”，则应用生成器（Application Genetical System，AGS）则可自动创建相应的应用软件。4GLs和应用生成器的开发研究可以使数据库应用系统的开发自动化和标准化，提高了效率和可靠性。
- 新一代数据库系统。传统的数据库系统虽然能有效地管理大量的数据，但只是对数据做简单的归整，运算也不复杂。随着计算机辅助设计技术和多媒体技术的发展，要求数据库具有更强的操作能力，这导致了数据操作语言和宿主语言的一体化趋势。目前，新型数据库系统的代表是面向对象数据库，已经推出了符合面向对象的数据模型的面向对象关系数据库系统。

数据库技术一直在不断地发展和演变，随着各种技术的突破，它必将使人类的信息数据处理跨入一个崭新的时代。

3.4.2 数据库管理系统的组成

数据库管理系统是一个十分复杂、庞大的软件。它可向用户提供专门的数据描述语言和数据操作语言，并能够解释和执行用户使用这种语言发出的命令和应用程序。

1. 数据描述语言

数据描述语言用来描述和定义数据库中的各种特征，主要包括以下几项。

- 描述数据的逻辑特征，即描述数据的逻辑结构，如数据文件名、记录类型名、数据

项名、数据的类型和长度及哪些数据项可以作为关键字来使用等。

- 描述数据的物理特征，即描述数据的存储结构，如说明系统中建立了哪些物理文件及其名字、类型存储设备、开始地址、组织方法等。
- 描述逻辑数据到物理数据的映射，即说明每个逻辑单位的数据存放在哪个物理文件中，以及从逻辑数据到物理数据的转换。
- 描述访问规则，即说明用户口令、存取权限及密码等。

不同数据库管理系统所提供的数据描述语言不完全相同，有的比较详细，有的比较简明，但都要求充分地描述数据的特征，且依赖于用户使用。

2. 数据操作语言

数据操作语言是用户对数据库中的数据进行操作的工具。对数据的操作主要包括以下几项。

- 打开和关闭数据库：当用户开始访问数据库时，先要用打开（Open）语句打开有关数据库文件；访问结束时，需用关闭（Close）语句关闭有关的数据库文件。
- 向数据库中增加新的数据：用户用存储（Store）语句和插入（Insert）语句等把新的数据添加到数据库中去。
- 修改数据中的数据：用户可以使用修改（Modify）语句来修改原有的数据，以反映数据的变化。
- 删除数据库中的数据：用户可以使用删除（Delete）语句将数据库中无保留价值的数据删除掉。
- 检索数据库中的数据：用户使用查找（Find）语句，从数据库中检索所需的数据，也可使用读取（Get）语句，把当前记录传送到用户指定的单元。
- 组织数据的报表输出：用户使用报表（Report）语句把检索到的数据按一定的格式输出。

不同数据库管理系统所提供的数据操作语言的语句种类和形式有所不同，但都要求能充分地描述对数据库中数据的操作，且便于用户使用。

3. 数据库管理系统的结构

数据库管理系统犹如一个数据描述语言和数据操作语言的解释器。其主要任务是解释并执行用户使用数据描述语言和数据操作语言发出的命令或用他们编写而成的应用程序。为完成这一任务，数据库管理系统通常由如图 3-7 所示的总控制程序及 6 个程序模块组成。

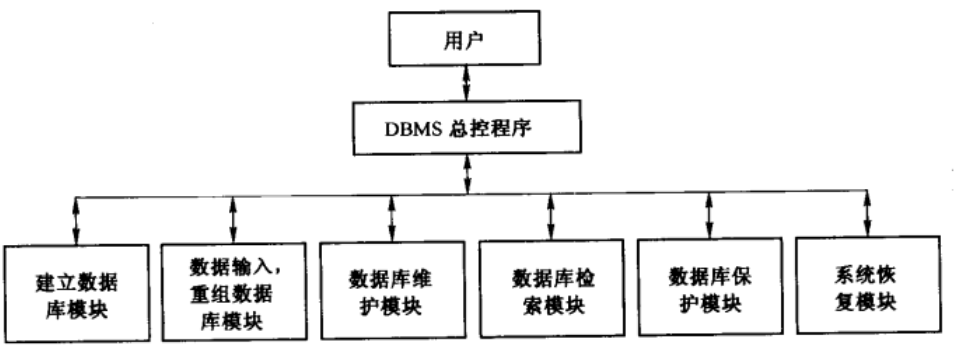


图 3-7　数据库管理系统模块结构示意图

下面对其功能做一简要说明。

- 总控程序。该总控程序负责协调各模块的工作，包含数据描述语言和数据操作语言的语言处理程序，对用户使用数据描述语言、数据操作语言书写的命令及应用程序进行分析，判明各命令的性质后转向各功能模块。
- 建立数据库模块。它根据用户对每个数据项的描述、类型定义及宽度等信息，建立数据库的结构。
- 数据输入和重组数据库模块。它按照定义好的数据库结构，自动提示用户输入相应的数据，并将大批数据存储到外存中，完成向数据库输入数据的工作，当数据库性能变坏时，它负责对数据库进行重新组织。
- 数据库维护模块。它按照用户的要求，从数据库中找出有关数据，并执行修改、插入及删除等操作。
- 数据库检索模块。它按照用户发出的检索操作的要求，从数据库中找出所需的数据，并通过显示或打印设备提供给用户。
- 数据库保护模块。在执行数据库操作之前，由该模块核对保密规定，如检查口令、审定存取权限等。
- 系统恢复模块。当系统发生故障时，由该模块将数据恢复到可用的状态。

3.4.3 数据库管理系统的分类

根据数据库的数据模型的数据设计方法不同，数据库管理系统可分为层次模型、网状模型及关系模型三种类型。

1. 层次模型

层次模型是以记录类型为结点的树型结构，如图 3-8 所示。图中包含 6 个记录类型。其中，“系”结点无上层结点，称为“父结点”。它有三个数据项，即系号、系名、系主任名。其他的结点都有一个父结点，且只有一个父结点。不同结点包含有不同的数据项。这样，每一个根记录与它从属的记录类型中的有关的记录一起便可以构成一棵树。它包括了该层次模型中的所有当前数据。

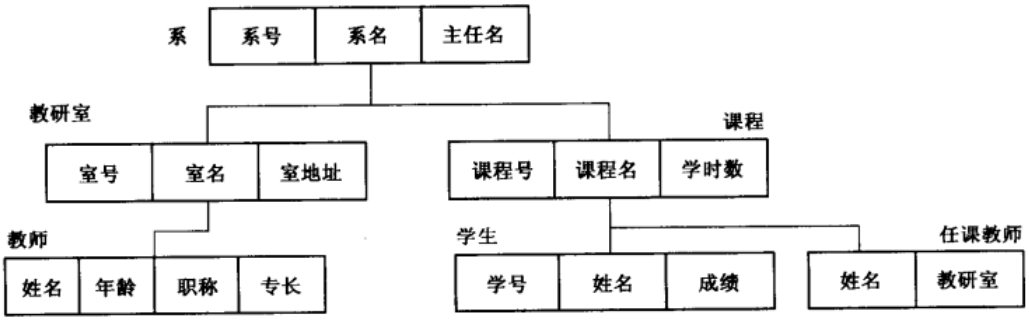


图 3-8 层次模型

由上可知，层次模型的记录类型间只有简单的层次联系，且满足下列条件：

- 有一个记录类型没有父结点；

- 其他记录类型有且只有一个父结点，即一个父记录可对应多个子记录，而一个子记录只对应一个父记录。

在层次模型中，必须按照从根开始的某条路径提出询问；否则，就不能直接回答。例如，要从如图 3-8 所示的层次模型中询问某科的任课老师的职称是什么？就不能直接回答，因为在“课程”与“教师”两个记录类型之间无直接通路。此时应该用如下的方式询问，即“某门课的任课教师是谁?”回答后，再问“此教师的职称是什么?”

数据模型采用层次模型的数据库具有结构简单、层次清楚及易于实现等优点，故在国际上较为流行。IBM 公司研制的最早的大型的数据库系统 IMS 便是层次数据库管理系统的典型代表。它应用于 IBM4300 系列计算机中。

2. 网状模型

若将层次模型的两个条件改成：

- 有一个以上的记录类型没有父结点；
- 至少有一个记录类型多于一个父结点。

则用这种网络结构表示记录类型之间联系的模型被称为网状模型，如图 3-9 所示。图中，“教师”和“学生”两个结点都没有父结点，而“成绩”结点却有两个父结点。

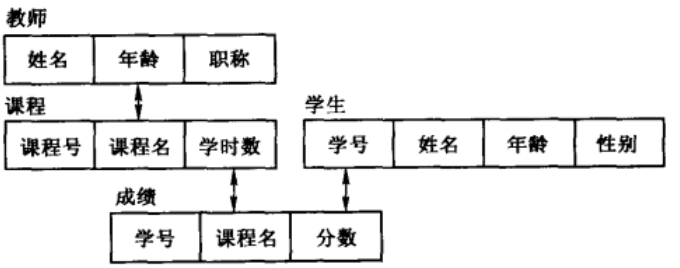


图 3-9　网状模型

就本质而言，网状模型和层次模型都是用指针来实现两个记录之间的联系的。其差别在于网状模型中的连线更加复杂，从而使数据结构更加复杂，应用不便，但其更接近实际。

网状模型是出现较早的一种数据库的设计方法。其典型代表是美国 CODASYL 组织研制的 CODASYL 系统。

3. 关系模型

关系模型的基本思想是把事物与事物之间的联系用二维表格的形式来描述。一个关系可以看作是一张二维表：表中的每一行是一个记录，在关系中称为一个元组；表中的每一列是一个字段，在关系中称为一个属性。第一行是各字段“型”的集合，构成一个框架，即为记录的“型”。其余各行则是各个记录的值。

例如，描述高校情况的数据库，由现实的情况可知：一个系有若干个学生，则一个学生只属于一个系；一个系只有一个系主任；一个学生可选多门课程，每门课程可由若干的学生选；每个学生选一门课程就有一门成绩。

对于上述情况，可以建立三个关系：

- 学生与所在系的关系，其数据项是学号、学生名及系名；
- 学生、课程与成绩的关系，其数据项是学生、课程及学习成绩；

↘系及其负责人的关系，其数据项是系名、系主任名。

关系模型可用离散数学中的关系代数来描述。因此，关系数据库管理系统能够用严格的数学理论来描述数据库的组织和操作，且具有简单灵活、数据独立性高等特点，故是目前应用十分广泛的一种数据库管理系统。关系模型是在 1970 年由 IBM 公司的 Codd 在其论文“一个通用关系式数据库系统的模型”中首先提出的。此模型后经其本人加以改进，现已被许多人认为是一切数据库系统的未来。

最后需指出，同一客观事物可以用上述三种不同的数据模型来描述，并且可以相互转换。例如，若对网状模型加以限制，就能变为层次模型；若将网状模型中的每一个记录类型及其相互之间的从属关系变为关系框架，则可将网状模型变成关系模型。然而，这些模型的数据存储和访问方法是不同的，用层次和网状模型设计的数据库系统是通过指针查找数据的；用关系模型设计的数据库系统则是通过查表查找数据的。加快查表速度比改进指针速度更有潜力，这一点也促使当前数据库系统的研究方向转向关系模型的研究。

3.4.4 常用关系数据库管理系统

尽管关系型数据库系统比层次型和网状型数据库系统的起步较晚，但由于它具有较完整的理论系统及使用灵活方便等特点，故后来居上。目前，在计算机上运行的数据库管理系统几乎全是关系型的，在我国运行较多的关系型数据库管理系统有 INGRES、INFORMIX、UNIFY、ORACLE、FoxPro 等。下面对其中主要几种数据库管理系统的概况做一介绍。

1. INGRES 数据库管理系统

该系统由美国加州大学伯克得分校电子研究实验室于 1975 年推出，原应用于 PDP-11 小型计算机中。该系统所使用的操作系统是 UNIX，所用语言是非过程化的 QUEL 语言，与外部的接口语言是 C 语言。该系统在某些微型计算机上已开发出汉化版本，并得到应用。其主要特点是：语言非过程化；无并发控制；无新型开发工具；效率较低。

2. ORACLE 数据库管理系统

该系统由美国 ORACLE 公司于 1980 年推出，可应用于大型计算机、小型计算机和微型计算机（如 IBM 的 PC/XT/AT）。该系统所使用的操作系统是 UNIX 或 DOS，所使用的语言是标准的 SQL 语言，与外部的接口语言是 COBOL、PL/1 和 Pascal 语言等。该系统已汉化，在大、中、小型计算机上都配有相兼容的版本。其主要特点是：有完善的软件开发工具；高度的兼容性；异种机数据共享；有数据保护功能。

3. INFORMIX 数据库管理系统

该系统是由美国关系数据库公司于 1981 年推出的，可应用于小型计算机（如 VAX 机）和微型计算机。该系统所使用的操作系统是 UNIX 或 DOS，所使用的语言是 RDSQL，与外部的接口语言是 C 语言。该系统已开发出适应各种机型的汉化版本，应用较广泛。其主要特点是：很强的完整性保护；模块组合结构；物理数据可分布。

4. FoxPro 数据库管理系统

FoxPro 是 Fox Software 公司自 FoxBase+1.12，2.00 和 2.10 等软件之后推出的 Fox 数据库应用系列的又一杰出产品。它与 DBASEⅢ、FoxBase 完全兼容，并提供一个编译型的集成环境供用户使用。FoxPro 具备了一个优秀微型计算机数据库管理系统所应该具备的几乎所有的特点。其主要特点概括如下：

- 具有一个良好的图形界面和窗口功能，反应敏捷，且速度很快；
- 具有较完善的真编译功能，可编译为直接在操作系统下运行的执行文件；
- 具有一个内部报表生成器，比独立的报表生成器更易于使用且功能更强；
- 采用“Rushmore（快速查找）”的优化技术，使它比任何其他微型机数据管理系统的查找速度都快，当数据库越大时，其速度优势越明显；
- 采用“Compact（紧缩）”类型的索引方法，使建立的索引文件长度缩短 20%以上，这对于大型数据库来说是十分可观的；
- 提供了高级外部程序接口 API，可以方便地实现与 C 语言和汇编语言的连接；
- SQL（结构化查询语言）是日趋流行的一种标准关系型数据库管理系统语言。FoxPro 的关系举例检索（RQBE）功能就是通过 SQL 的 Select 命令处理的。其数据检索方便灵活，但目前的 FoxPro 还不全面地支持 SQL。

FoxPro 所具有的强大功能及优异的特点，使它一经问世就受到了广大软件开发人员和计算机用户的高度重视。FoxPro 的应用前景十分美好。

小结

软件系统是计算机系统的一个重要组成部分，了解和掌握计算机软件系统是十分必要的，而且也是十分重要的，对学生今后学习计算机专业课起着指导作用。

本章主要介绍了计算机软件系统的基本概念及其分类、程序设计语言、操作系统、数据库系统等方面的知识。

在计算机软件系统基础知识部分，主要应掌握计算机软件的基本概念和分类，了解系统软件和应用软件的基本构成和作用。

在计算机程序设计语言部分，主要介绍的是计算机机器语言、汇编语言及高级语言的各自特点和各种高级语言的应用领域。

在操作系统部分，主要介绍的是操作系统的作用和分类及各种操作系统的特点，并重点介绍了操作系统中的进程管理、存储器管理及中断技术。

在数据库管理系统部分，主要介绍的是数据库的基本概念和数据库系统的基本构成，并讲述了数据库管理系统中的层次模型、网状模型、关系模型的各自特点及目前在我国广泛使用的关系数据库系统。

习题

1. 什么是软件？它由几部分组成？
2. 请举出你所见到的应用软件系统。
3. 什么是机器语言？它的优、缺点是什么？
4. 试述计算机语言的分类及它们的主要特点。
5. 什么是操作系统？它可分为哪几类？
6. 什么是数据库、数据库管理系统及数据库系统？
7. 什么是进程？最基本的进程状态有哪些？举例说明进行状态转换的典型原因及实现方法。

8. 什么是中断？试述中断响应过程。
9. 试述页式存储管理的基本原理。
10. 什么是虚拟存储技术？它的基本思想是什么？
11. 简述数据库系统的特点。
12. 数据库管理系统是怎样组成的？
13. 什么是数据操作语言？列举主要操作语句及其功能。
14. 简述数据管理系统的三种模型的特点和区别？
15. 常用的关系数据库管理系统有哪些？

第 4 章　计算机网络系统

本章要点

- ☑ 计算机网络的发展历程
- ☑ 计算机网络协议
- ☑ 计算机网络分类
- ☑ 计算机局域网的组成和连接设备
- ☑ 计算机局域网实例

4.1　计算机网络概述

计算机网络把孤立的各个计算机连接起来，实现了众多计算机硬件和软件资源的共享。可以说，计算机网络是计算机发展的必然。

4.1.1　计算机网络的发展历程

今天的互联网实际上是 50 多年前美国大搞军事情报战的产物。20 世纪 50 年代，美国与前苏联军备竞赛时期，美国军方担心前苏联的飞机绕道北极来空袭，于 1951 年资助麻省理工学院成立了著名的林肯实验室，开始“远距离预警系统”的研究。该系统是一种中央控制式网络结构。它的目的是：第一，采集从各雷达站来的信息；第二，通过计算判断是否有敌机来犯；第三，将防御武器对准来犯之敌。该系统于 1952 年开始投入使用，曾为美国搜集军事情报和协调各军事部门的作战立下了汗马功劳。

随着前苏联 1957 年 10 月 4 日和 1957 年 11 月 3 日两颗人造地球卫星分别发射成功，美国军方感到了极大的威胁。其原因是前苏联的卫星飞到了美国的上空才被美国发现。美国因此对其拥有的中央控制式网络产生了怀疑。1958 年，当时的美国总统向国会提出了建立国防部高级研究计划署（Defense Advanced Research Projects Agency，DARPA）的建议，希望通过该机构的努力，确保美国在军事上对前苏联的均势或优势。

1962 年，美国军方向当时的美国总统提出了一份建议，强调美国当时的网络布局存在着严重隐患，即前苏联的导弹只要摧毁该网络的中心控制部分，则整个网络就会瘫痪；美国军队通信联络的网络化程序越高，受到的破坏也越强烈。这一问题引起了美国军界的重视，随即 DARPA 着手组织改进网络结构的安全性，以确保美国军队的网络系统在遭受打击后，仍能够保证各部门之间通信联络的畅通。

DARPA 经过长时间的研究和论证后得出，可以设计出一种分散的指挥系统。它由许多指挥点组成。当部分指挥点系统遭到破坏之后，其他指挥点系统仍能正常工作，并且这些指

挥点可以绕过那些遭到破坏的指挥点保持联络。为了实施这一计划，DARPA 建立了 ARPANET 网络。该网络将加利福尼亚大学洛杉矶分校等几所大学的计算机主机连接起来。各个结点的大型计算机采用分组交换技术，通过专门的通信交换机和专门的通信线路相互连接。该网络开始时只有 60个结点，主机有 100 多台，地域覆盖美洲大陆，还可通过卫星与夏威夷、欧洲等地区相联，这就是最早的计算机网络。

ARPANET 网络的特点是资源共享、分散控制、分组交换，采用专门的通信控制处理机、分层的网络协议。

许多大学、研究机构、军事基地及政府实验室中的研究人员经常使用 ARPANET 网络交换文件、发送电子邮件和进行远程登录。1975 年，DARPA 将网络转给了美国国防通信局 DCA（Defense Communication Agency）。DCA 将 ARPANET 网络变成国防数据网络 DDN（Defense Data Network）的一部分。在 Internet 网广泛使用后，ARPANET 网络的发展开始下滑。

认识到计算机网络的重要性后，美国国家科学基金会 NSF（National Science Foundation）组织建立了 NSFNet 网络。NSFNet 网络的大小和容量不断增加，最后形成了现在广泛使用的 Intenet 网中的主干网。

NSF 的 Intenet 框架由三个层次构成：第一层次是国家主干网；第二层次是一组覆盖小地理区域的中级网络；第三层次是一组“校园”网络。中级网连接到国家主干网上，校园网连接到中级网络上，计算机连接在校园网上。

4.1.2 计算机网络的概念

如果要给计算机网络下一个定义，则可以这样描述：计算机网络是由用通信线路连接的许多自主工作的计算机构成的集合。这里需要强调两点：首先，“通信线路”除了为铜导线外，还包括无线信道，如微波、毫米波、红外线及电磁波等；其次，“自主工作”是指不存在主从关系，是指计算机脱离网络环境之后则可自成体系的计算机系统，即可以独立地完成工作。也就是说，网络中单独一台计算机有自己的操作系统，有自己的内存，有自己的 CPU。

4.1.3 计算机网络协议

为了实现计算机之间的通信，计算机之间必须遵循共同的规则，即网络通信协议。也就是说，计算机之间进行通信时必须以约定的协议进行。协议具体到计算机中则是一组实现规则的软件。它的功能是使相互连接的计算机之间按照协议准确、及时地传递数据。

1977 年，国际标准化组织 ISO 下设的第 16 分委员会 SC16 开始研究开放系统互联的标准。所谓开放系统，就是指任何系统只要遵循这个标准进行设计，就可以与其他任何遵循同一标准的系统互相连接并进行通信，这就确保了各个厂商的网络产品能相互兼容。SC16 于 1979 年完成了基于分层概念的网络协议开发模型称这一模型为开放系统互联参考模型 OSI/RM（Open System Interconnection/Reference Mode）。同年底，国际电报电话咨询委员会 CCITT（Consultative Committee，International Telegraph and Telephone）认可并采纳了这一国际标准（称为 X.200）。

OSI 共分为 7 层。它们是物理层、数据链路层、网络层、传输层、会话层、表示层及应用层。当网络上的计算机需要发送数据时，就将发送的数据下传一层，再加上该层的标识

(俗称打包)，这样逐层下传，直到物理层，然后将数据通过传输介质发送给对方。对方接收到数据后，进行反方向拆开（俗称解包），然后逐层上传，直到应用层。其数据传送的 OSI 参考模型如图 4-1 所示。

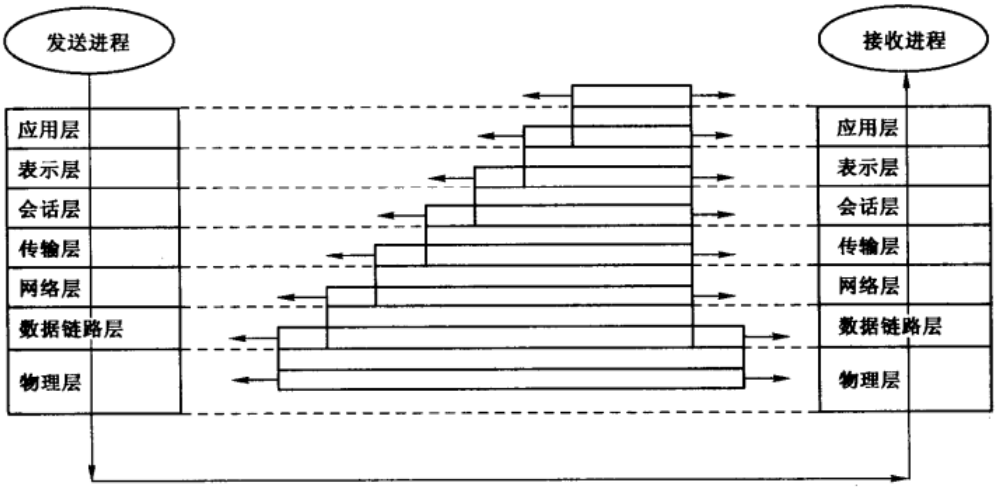


图 4-1　数据传送的 OSI 参考模型

需要说明的是，OSI 只是一个标准，而不是一个具体的协议。所谓标准，是指制造商在实现协议时应该遵守的规定，或标准化组织建议制造商在实现协议时遵守的规定。而协议是指制造商在其产品中具体实现的通信规定。标准是制定协议的原则或框架。

计算机网络的协议有很多，最著名的有 TCP/IP（Transmission Control Protocol/Internet Protocol——传输控制协议/网间网协议)。它是 Internet 上使用最多的协议。另外还有 NetBEUI（NetBIOS Extended User Interface——NetBIOS 扩展用户接口)、IPX/SPX（Internetwork Packet eXchange/Sequences Packet eXchange——网络包交换 /顺序包交换，Novell Netware 局域网规程)、Apple Talk（苹果公司的计算机局域网协议)、DECNet（Digital Equipment Corporation Net——美国数字设备公司网络规程)、ATM（Asynchronous Transfer Mode——异步传输模式)、x.25 等协议。

4.2　计算机网络的分类

计算机网络可以按照范围大小、拓扑结构和交换方式进行分类。

4.2.1　按网络的范围大小分类

按网络的范围大小，计算机网络可分为局域网 LAN（Local Area Network)、城域网 MAN（Metropolitan Area Network）和广域网 WAN（Wide Area Network)。

1. 局域网 LAN

所谓局域网，是指覆盖范围受到限制的网络，一般是在 10 m～10 km 的范围内。例如，同一个房间、同一个建筑物、同一个园区等。通常的校园网、企业网、公司内部网等都属于局域网的范畴。

2. **城域网 MAN**

城域网是大型的 LAN，是指一个城市大小范围的网络，一般是在 10 km 到几十千米的范围内。当然，城市有大有小，西安是一个城市，北京是一个城市，纽约也是一个城市。这里强调的只是一个相对的概念。城域网在网络术语中使用的不多。

3. **广域网 WAN**

广域网的范围指距离超过 100 km 的网络。广域网可以指同一个国家或同一个洲的计算机网络，甚至可以把全球范围的计算机网络看成一个广域网。

按网络的覆盖范围大小分类只是一个相对的概念，并不是绝对的。也就是说，按网络的覆盖范围大小分类时，地域界限范围的划分不是十分严格的。随着网络技术的发展，计算机网络延伸的范围会越来越广泛，使用的技术也越来越复杂，则按上面的分类方法有时很难划分。按照现阶段的实际情况，有些学者将计算机网络分为工作组网、校园网（社区网）、企业网（公司网）及全球广域网。

广域网是由资源子网和通信子网组成的，如图 4-2 所示。资源子网的功能是以信息共享为目的的，一般由若干台主机构成。通信子网的功能是将信息从一台主机传递到另一台主机。通信子网由两个不同的部分——传输线路部分和交换单元部分组成。传输线路也称为线路、信道或干线。其功能是在主机之间传送数据。交换单元是一种特殊的计算机，用于连接两条或更多条传输线路。当数据从输入线到达时，交换单元必须为数据选择一条输出线以便传送它们。通常将交换单元称为路由器（Router）。传输线路和路由器（不包括主机）的集合就组成了通信子网。

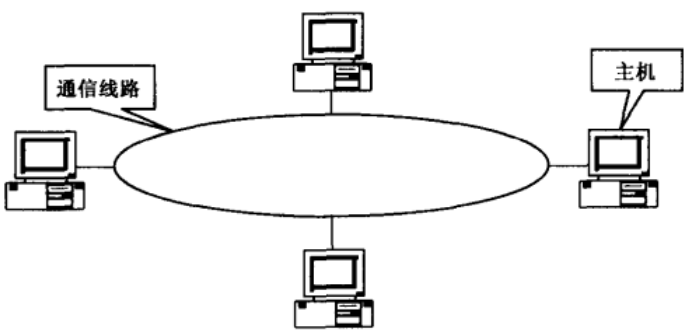


图 4-2 通信子网与资源子网

广域网采用的设备是大量的电缆和电话线，每一条都连接一对路由器。如果前面的路由器要发送信息，则其后面相邻的路由器在接收到信息之后，首先要进行校验，在正确接收之后进行保存，然后当输出线路空闲时再向下一个路由器发送，就像接力棒一样从一个节点向下一个节点传送，直至信息到达目的地。使用这种原理的子网被称为点到点、存储转发或分组交换通信子网。

在计算机网络中，通信子网是由网络经营者拥有的路由器和通信线路组成的集合。计算机网络中的通信子网就像电话系统中的通信网络。电话系统由许多个电话交换子系统组成。各电话交换子系统之间由线路连接。每个电话交换子系统再连接若干个电话用户。这些电话交换设备和通信线路属于电话系统的通信网络。而电话机本身（与之相对应的是计算机网络中的主机）不是通信网络的一部分。

4.2.2　按网络的拓扑结构分类

通常，把计算机网络上的通信设备及其连接的计算机统称为网络工作站或网络结点，而把网络结点的位置及其互联的几何布局称为网络的拓扑结构。按网络的拓扑结构可以将计算机网络分为总线、星状和环状网络。

1. 总线网络

总线网络结构是用一条通信线路（总线）将每一台主机都连接起来的结构。这种结构属于共享、广播方式的通信。总线网络在总线的两端必须设置电阻器，以防止信号的反射。总线网络的特点是：

- 一般使用同轴电缆进行连接；
- 适用于少量的计算机连接，一般少于 20 台；
- 网络的稳定性差，任一点的断开都会使网络瘫痪；
- 主要用于低速带宽，即 10 Mbps 以下带宽的共享网络。

总线网络的拓扑结构如图 4-3 所示。

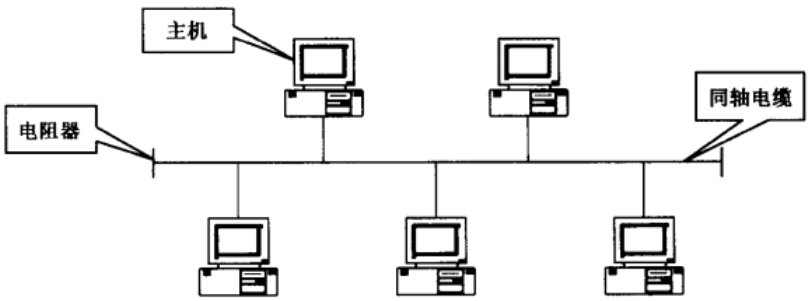


图 4-3　总线网络的拓扑结构

2. 星状网络

星状网络结构将所有的主机都连到集线器或交换机上，任何两台主机之间的通信都要经过集线器或交换机。星状网络的特点是：

- 集线器或交换机可以进行级联，但级联数量最多不能超过 4 级；
- 工作站的接入或退出不影响网络的正常工作；
- 可以满足多种带宽（如 10 Mbps、100 Mbps、1000 Mbps）的要求。

星状网络结构是局域网中经常使用的一种结构。其拓扑结构如图 4-4 所示。

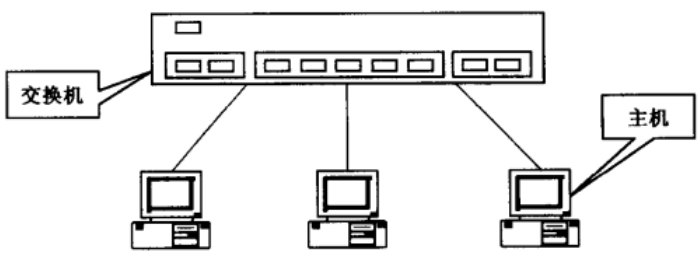


图 4-4　星状网络的拓扑结构

3. 环状网络

环状网络结构是将所有的主机均连接在一个封闭的环中，信号依次通过所有的主机，最后再回到起始主机。每个主机是否接收此信息，取决于信息的目标地址。在环状网络中，每个主机相当于一个中继器，在接收到信号后可以恢复信号原有的强度，从而可以保证信号的传输质量。环状网络的拓扑结构如图 4-5 所示。

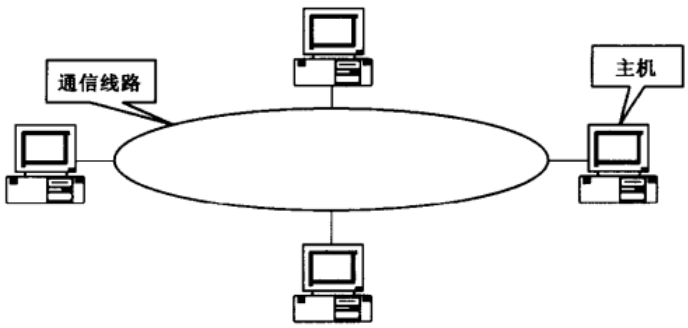


图 4-5 环状网络的拓扑结构

环状网一般在中、小型局域网中很少使用，主要用在广域网中。

4.2.3 按网络的交换方式分类

交换结点转发信息的方式被称为交换方式。按交换方式可以将计算机网络分为电路交换、报文交换及分组交换。

1. 电路交换

电路交换也称为线路交换。电路交换方式是把发送方和接收方用一系列链路直接连通。例如，目前使用的电话交换系统就是采用这种交换方式。

电路交换的特点是数据传输前需要建立一条端到端的连接，建立连接需要等待较长的时间。由于通路专用，因而不会有干扰和冲突，不会有延迟，一旦接通后，数据的惟一延迟是电磁信号的传播延迟。电路交换适用于传输大量的数据，当传输少量数据时，效率比较低。

2. 报文交换

报文交换不需要预先建立通路。发送数据时，数据被放到第一结点（即路由器），检查是否有错。若无错误，则发送出去，然后一级一级地中转。使用这种技术的网络被称为存储－转发（store-and-forward）网络，如电报系统、E-mail 等都属于报文交换。

3. 分组交换

分组交换是基于数据分组（Packet）的一种网络交换方式。在进行分组交换时，发送结点要对传送的数据进行分组，并对各个分组进行编号，然后加上源地址和目的地址及约定的头和尾的信息。这一过程叫信息的打包，最后才将分组发送出去。

在分组交换中，一个结点在后续分组到来之前，前面的分组可以先转发出去，这样就可以减少延迟和提高吞吐量。正是由于这个原因，计算机网络常常采用分组交换。

依据在网络中的数据传输方式，又可将分组交换分为数据报和虚电路。

（1）数据报

数据报类似于报文交换，每个分组在网络中的传播路径完全是由网络当时的状况决定

的。因而，每个分组都有完整的地址信息，如不出意外的话，则一般都可到达目的地。在数据报交换方式中，发送时要将数据进行分组，接收时分组的顺序可能与发送时分组的顺序不一致。因此，目标主机要对收到的分组重新进行排序。

(2) 虚电路

虚电路类似于电路交换。这种方式要求在发送端和接收端之间建立一条所谓的逻辑连接。与电路交换不同的是，逻辑连接的建立并不意味着别的信息传输不能使用这条线路，即此种数据传输方式具有线路共享的特征。

4.3 局域网

局域网在计算机网络中占有重要的地位。局域网是构成广域网的基础。

4.3.1 局域网的组成

计算机局域网一般由服务器（Server）、工作站（Workstation）、外围设备及通信协议组成。

1. 服务器

服务器是整个网络系统的核心。服务器为网络用户提供服务并管理整个网络。服务器的类型有文件服务器、通信服务器、邮件服务器、备份服务器及打印服务器等。

小型局域网中的服务器一般提供文件和打印两种服务，并且通常将文件和打印服务集中到一台计算机上进行。服务器是一台计算机，只是一般要求这台计算机的性能较高；否则，整个局域网的使用效率将很低。目前，中、小型局域网中一般使用性能较高的微型计算机作为服务器。

2. 工作站

工作站是指连接在网络上的计算机，不同于服务器。工作站只是一个接入设备。它的接入和离开并不对网络系统产生影响。工作站也被称为用户或客户机。

3. 外围设备

外围设备分为传输介质和连接设备两类。传输介质包括同轴电缆、双绞线和光纤。

(1) 同轴电缆

同轴电缆的中心有一条铜导线，外面包一层绝缘体，在绝缘体外再包一层网状铜导体用于屏蔽电磁波的干扰。同轴电缆的最外层又是一层绝缘体。同轴电缆一般用于总线型网络结构。同轴电缆主要有基带同轴电缆和宽带同轴电缆两种。基带同轴电缆为 50 Ω，主要用于 4 kHz 以下频率的数据传输；宽带同轴电缆为 75 Ω，主要用于 4 kHz 以上频率的数据传输。

(2) 双绞线

双绞线是一种最常使用的传输介质。目前使用的双绞线内部有 8 根铜导线，每根铜导线外的绝缘层都使用不同的颜色以示区别。每两根铜导线缠绕在一起，可防止信号干扰。双绞线分为非屏蔽双绞线和屏蔽双绞线两大类。其中，非屏蔽双绞线是经常使用的线型。

双绞线一般用于星状网络结构的布线。每条双绞线的两端都安装有 RJ-45 连接器（也称为水晶头）。RJ-45 连接器可以插到网卡和集线器或者交换机的端口上。单根双绞线允许的最大长度为 100 m。若要延伸，则最多可以级联 4 级集线器或者交换机。因此，在星状网络

中，两台工作站之间的最远距离为 500 m。

(3) 光纤

光纤即光导纤维，是利用光在两种介质的界面上发生全反射的原理制成的一种传输介质。光纤中传输的是光信号。光纤一般使用玻璃材料，目前已经出现了使用高级塑料的光纤。光纤可以进行高速度、长距离、大容量的数据传输。光信号在传输时有信号衰减，为了保证传输质量，每隔一段距离需设置中继机以恢复信号质量。

光纤可以用 1 Gbps 的速度一次传输 50 km 左右而不使用中继机。光纤的两端安装有光纤收发器，用于光电信号的转换和光信号的收发。

将一束光以一定的角度进入光纤称为一个模式（Mode）。按模式可以将光纤分为单模光纤和多模光纤两种。多模光纤的缺点是线芯比较粗，传输的速度低、距离短、性能差，但成本低。如果光纤的直径减少到一个光波波长大小的时候，则光纤如同一个波导，光在其中没有反射而沿直线传播。这种光纤被称为单模光纤。单模光纤的优点是线芯比较细，传输的速度高、距离远、频带宽、容量大，但成本高。

光纤目前主要用于交换机到交换机的连接及交换机到服务器的连接。局域网中一般使用 50 μm/125 μm（内径/外径）、62.5 μm/125 μm 等规格的多模光纤及 8.3 μm/125 μm 等规格的单模光纤。实际使用的光缆中一般包含有多根光纤。

除了上面三种有线局域网传输介质外，局域网传输介质还包括无线传输介质，如电磁波、无线电、微波、红外线、毫米波及激光等。在无法铺设或者不允许铺设电缆的环境中，可以使用无线传输介质。

4.3.2 局域网的连接设备

局域网的连接设备包括网卡、集线器、交换机（Switching HUB）等。

1. 网卡

网卡是网络接口卡的简称。网卡是计算机局域网中最重要的和必须的连接设备。网卡插在计算机的主板上，与主板总线连接。工作站是通过网卡接入网络的。网卡是工作在数据链路层的设备。它的功能是自动接收和发送数据。接收数据时，网卡将数据解包后传送到该工作站中；发送数据时，网卡将数据打包后发送到网络中去。

网卡的分类方法有以下几种。

(1) 从不同工作对象的角度考虑，网卡有普通工作站网卡、服务器专用网卡、便携式计算机网卡及无线局域网网卡。

(2) 从速度方面考虑，网卡有 10 Mbps 网卡、10/100 Mbps 自适应网卡、100 Mbps 网卡及 1000 Mbps 网卡等。

(3) 从网卡插入的总线类型角度考虑，网卡有 PCI 总线网卡、ISA 总线网卡等。

(4) 从连接方式的角度考虑，网卡有光纤接口、RJ-45 连接器接口（双绞线使用）、BNC 连接器接口（细缆使用）及 AUI 连接器接口（粗缆使用）等。

2. 集线器

集线器是一种管理网络的工具。集线器位于星状网络的根结点，工作在物理层，是一个共享式设备，相当于一个中继器，起到信号的放大再生作用。

集线器的分类方法主要有以下几种。

（1）从速度角度来分，集线器分为 10 Mbps、10/100 Mbps 自适应、100 Mbps、100/1000 Mbps 自适应、1000 Mbps 等；

（2）从配置形式来分，集线器分为独立式、模块式及堆叠式等；

（3）从端口数目来分，集线器分为 8 口、16 口、24 口及 48 口等。

3. 交换机

集线器仅仅起到信号的放大再生功能。相比之下，交换机在功能上要强得多。交换机具有网络交换的功能、可以进行网络管理及可以自动检测网络端口的速度等。目前，局域网中使用的一般是交换机。有时将交换机也称之为交换式集线器。

4. 通信协议

上述网卡、集线器和交换机只是实现了局域网硬件设备的连接。除了硬件设备的连接外，局域网通信还要有通信协议。通信协议在 4.1.3 节中已做过讨论。两台计算机之间进行通信时，必须使用相同的通信协议。局域网的协议一般有 NetBEUI、TCP/IP 和 IPX/SPX 三种。

4.3.3　局域网的结构

局域网的常见结构有对等式网络结构、专用服务器结构和客户－服务器结构。

1. 对等式网络结构

对等式（Peer to Peer）网络是指网络中不需要专用的服务器，每一台接入的工作站既是服务器，也是工作站，拥有绝对的自主权。可用于对等式网络的操作系统有 DOS、Windows 9x、Windows NT、Windows 2000 等。

2. 专用服务器式结构

专用服务器（Server Based）式结构是网络中必须要有一台专用文件服务器，工作站与工作站之间通信时必须通过文件服务器进行。NetWare 网络操作系统就是典型的专用服务器结构。这种结构有利于网络的严格管理，保密性好，可靠性高，但工作站上的资源无法实现共享。

3. 客户－服务器结构

客户－服务器（Client/Server）结构也被称为主从式结构。它解决了专用服务器结构中存在的不足。客户端既可以与服务器端进行通信，同时客户端之间也可以不需要经过服务器直接通信。WindowsNT server 和 Windows 2000 Server 是典型的客户－服务器结构操作系统。客户－服务器结构效率比较高，可以实现工作站资源的共享，但安全性比较差。

值得注意的是，局域网的网络结构与它的拓扑结构是没有关系的。拓扑结构强调的是网络中工作站及其设备的连接方式，而网络结构强调的是网络内部数据的传输方式。

4.4　计算机局域网实例

现以某厂计算机网络管理信息系统为例，说明计算机局域网（Local Network）的构成，并对计算机网络有一较为具体的认识。图 4-6 是该系统的硬件配置图。

它包括如下几部分。

- 文件服务器：IBM SEVER X235 8671-21X。

- 结点机（网络工作站）：P4 兼容机。
- 网络适配器（网卡）：TP-Link 3239D。
- 网络拓扑结构：星状。
- 调制解调器（Modem）：蓝科 L2000。
- 其他：5 类 UTP；RJ-45；TP-Link TL-SF1016。

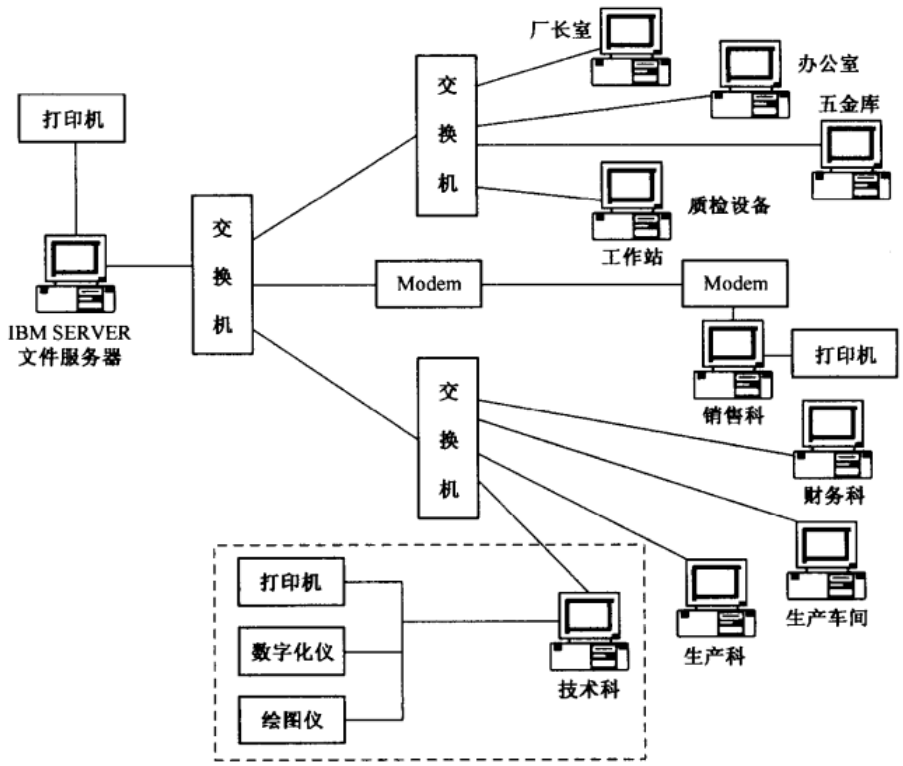


图 4-6 局域网实例

由如图 4-6 所示可知，这是一个小型的计算机局域网。网上共有 9 台 P4 微型机，1 台 IBM SERVER X235 8671-21X 服务器，通过双绞线与交换机相联。可见，该网络的拓扑结构是星状的。网上文件服务器是一台 IBM SERVER 服务器，它用做网络的公共数据库。任何一个工作站都可将数据存入文件服务器，并从那里读取数据。图 4-6 中，“销售科”工作站是通过调制解调器与内部网络相联的，因为该工作站远离其他工作站（约 30 km），需借助于电话线来传送信息。

该网络由 Windows 2000 Server 网络操作系统来管理，采用广播通信方式。网络上任何一台计算机发出的报文都有可能被网上的其他计算机所接收，只是需要通过对报文中的“地址”分析，确定是真正发给自己的，才接收下面发来的报文，否则不予理睬。各工作站之间的通信必须遵守某种协议，如国际标准化组织（ISO）于 1984 年提出的“开放系统互联参考模型（OSI 协议）”。该协议适用于任何类型的计算机网络。对于如图 4-6 所示的计算机局域网，上述问题都已被 Windows 2000 Server 网络操作系统解决，用户只要会在网上安装和使用该操作系统就可以了。

随着计算机应用领域的不断扩大，计算机局域网的使用越来越普及。目前，较流行的计算机局域网有以太（ETHERNET）网、NOVELL 网、PLAN4000 网、PCNET 网及 DLINK 网等。不仅如此，计算机网络正向网络互联的方向发展，以至于各种计算机网络可以相互联网，并可与大型的计算机网络联网。

小结

计算机网络是把孤立的各个计算机连接起来，可实现众多计算机硬件和软件资源的共享。可以说，计算机网络是计算机发展的必然。

本章主要讲解了计算机网络的发展历程、计算机网络的分类及计算机局域网等方面的知识。

在计算机网络发展历程中，主要介绍的是计算机网络的发展历史、计算机网络及计算机网络协议的基本概念。

在计算机网络部分，主要介绍的是计算机网络的分类方法及所形成的各类计算机网络。

在局域网部分，主要介绍的是计算机局域网的基本组成及计算机局域网的基本结构，重点应掌握局域网中的连接设备，并通过实例介绍了计算机局域网的基本组成。

习题

1. 什么是计算机网络？为什么要联网？
2. 开放系统互联参考模型分为哪几层？各层的作用是什么？
3. 什么是协议？其作用是什么？
4. 计算机网络按地理范围分为哪几种？
5. 计算机网络拓扑结构有哪几种？各自有什么特点？
6. 按交换方式分，计算机网络有哪几种？
7. 计算机局域网由哪几部分组成？
8. 总线、星状、环状局域网各有什么特点？
9. 局域网经常使用的协议有哪些？
10. 计算机网络的有线传输介质有哪几种？哪种传输速度最快？
11. 计算机局域网常用的连接设备有哪些？
12. 常见的计算机局域网的结构有哪些？

第 5 章　计算机的基本操作

本章要点

- ☑ Windows 98 基础知识
- ☑ Windows 98 下文件管理
- ☑ 中文版 Word 的基本使用
- ☑ 中文版 Word 的高级使用
- ☑ Excel 的基本使用
- ☑ Excel 的高级使用

5.1　Windows 98 操作

5.1.1　Windows 98 概述

1. Windows 98 的特点

Windows 98 操作系统是美国微软公司推出的视窗操作系统，优秀的图形界面、简单方便的操作方式，使其成为在个人计算机中应用最广泛的操作系统。Windows 98 的特点如下。

- 图形界面，简便的操作方式。Windows 98 提供了人性化的图形界面，使用户与计算机交流更加容易。它为用户提供了菜单、工具栏及任务栏等界面，使用户无需像使用 DOS 那样去记忆那些繁琐的命令，从而使操作简单易学。
- 支持长文件名。在 Windows 98 中，文件名的长度可以不超过 256 个字符，字符有大小写之分，文件名中可以包含空格。
- 多任务并行操作。允许多个程序同时运行，每个程序占据一个窗口，用户可以方便地在各个窗口之间交换数据。
- 即插即用。Windows 98 提供了即插即用的功能，只要用户将硬件插入计算机，系统就能自动识别设备，并引导用户安装驱动程序。
- 多媒体应用。对于多媒体的应用，图形画面更加逼真，操作方式同操作真正的设备一样简单。用户能够更加容易的使用多媒体设备和软件。
- 网络通信。用户可以非常方便地将自己的计算机连接网络，浏览网上信息，收发电子邮件等。

2. 鼠标与键盘操作

Windows 98 的操作可以通过键盘来实现，也可以通过鼠标来完成。键盘是计算机系统最基本的输入设备，只通过键盘也能完成对 Windows 98 窗口和菜单等的操作。如果想要充

分发挥图形用户界面的优点，那还是需要利用鼠标。

(1) 鼠标及其使用

鼠标是一种比手掌还小的图形输入装置。由于它总是通过一条电缆与计算机相连，形似一只拖着长尾巴的老鼠而得名。鼠标上一般有两个或三个按键（中间一个很少使用）。其左右两个按键分别被称为鼠标左键和鼠标右键。当鼠标工作时，显示器上会出现一个表示鼠标当前位置的光标。此光标被称为鼠标指针。如果用户移动鼠标，则显示器上的鼠标指针也会随之移动，这样就可以用来指示各种操作对象。当选定对象后，只需单击鼠标左键或右键即可完成相应的操作。

鼠标主要的操作如下。

- 指向：移动鼠标，使鼠标指针定位在某个具体目标上，为后续操作做好准备；
- 单击：轻快地单击左键，用于选择某一具体目标；
- 双击：连续两次快速单击鼠标左键，用于启动某个操作；
- 拖动：按下鼠标左键不放，并移动鼠标；
- 右击：轻快地单击鼠标右键，用于某些特定操作，如打开快捷菜单等。

需要注意的是，当进行双击操作时，在连续两次击键期间不能移动鼠标；否则，双击操作无效，而被认为是两次单击。

(2) 键盘操作

键盘是最基本的输入工具，当文档窗口或对话框中出现闪烁着的插入光标时，就表示可以用键盘输入字符。除此之外，键盘还可以用来进行窗口和菜单操作。

在窗口和菜单操作中，按下 Alt 键及菜单中带下划线的字母，则可选中相应的命令。也可以用键盘上的光标移动键移动亮条，并按 Enter 键选取相应的命令。

5.1.2　桌面简介

对于安装了 Windows 98 的计算机，不必使用特殊命令，只要打开电源即可进入 Windows 98。Windows 98 启动后的整个屏幕区域被称为桌面，如图 5-1 所示。在桌面底部的长形区域被称为任务栏。任务栏左端凸起的是【开始】按钮，右端凹下去的是【状态指示器】。

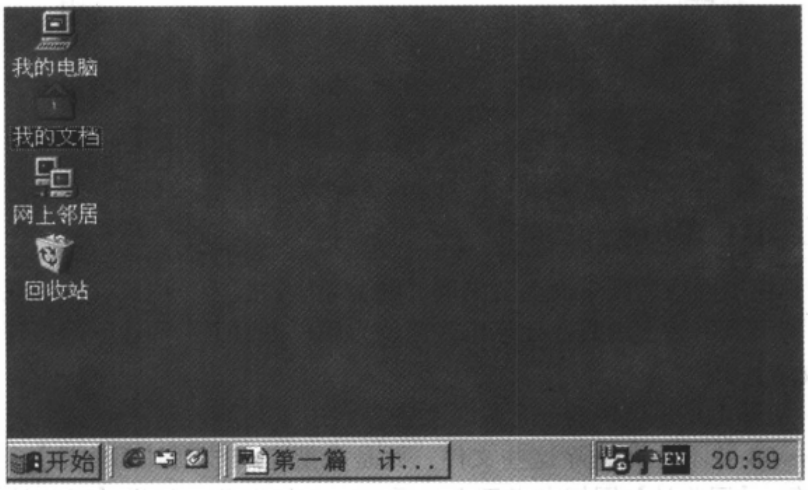


图 5-1　Windows 98 的桌面

1. 桌面上的图标

正如真正的办公桌上会摆放许多办公用品一样，在 Windows 98 桌面上也摆有一些代表程序或文件的小图形，称其为图标，如【我的电脑】、【回收站】、【网上邻居】等。用鼠标左键双击这些图标，则可打开相应的窗口。

Windows 98 的文件夹是用于存储程序、文档、快捷方式及其他子文件夹的地方。在大多数情况下，一个文件夹对应一块磁盘空间。文件夹的路径是一个地址，可告诉操作系统如何才能找到该文件夹。例如，许多 Windows 98 系统文件都存储在一个路径为 C：\ Windows 的文件夹中。

当打开一个文件夹时，它是以窗口的形式呈现在屏幕上的；关闭它时，则可收缩为一个图标。文件夹是标准的窗口，用来作为其他对象（如子文件夹、文件）的容器，并以图标的方式来显示所包含的内容。使用它可以访问大部分应用程序和文档，很容易实现复制、移动和删除对象。

下面简要介绍一下 Windows 98 桌面上的几个常用文件夹。

(1)【我的电脑】

【我的电脑】是一个很常用的文件夹。它可以用来查看计算机中的所有内容，以及对文件和文件夹进行复制、移动及删除等操作。

如果用鼠标左键双击【我的电脑】图标，则可打开如图 5-2 所示的窗口。其中包括【3.5 英寸软盘（A：）】、【（C：）】、【（G：）】及【打印机】、【控制面板】等图标。单击窗口右上角的【关闭】按钮即可以关闭该窗口。



图 5-2 【我的电脑】窗口

(2)【网上邻居】

如果用户正使用网络，则桌面上会出现【网上邻居】图标。用鼠标左键双击它即可浏览工作组中的计算机及网络上的其他计算机。这就为用户提供了一种访问网络资源的手段。

(3)【回收站】

【回收站】实际上是系统中的一个临时文件夹，用于存放被删除的对象。当用户删除文件时，Windows 98 并不立即删除这些文件，而是先把它们放到【回收站】中。如果错删，则可以进行恢复。【回收站】图标在桌面上显示为一个垃圾箱。如果【回收站】为空，就显

示一个空的垃圾箱；否则，显示一个装有废纸的垃圾箱图标。

除了以上常见图标外，根据系统所安装的软件和组件的不同，Windows 98 桌面上还可能出现一些其他的图标，如【收件箱】、【我的公文包】等。另外，用户也可以根据需要把自己常用的程序以图标形式放置在桌面上。

2. 任务栏和开始菜单

(1) 任务栏

Windows 98 桌面底部的条形区域被称为任务栏。当启动一个程序或打开一个窗口后，在任务栏上就会出现一个与之相对应的按钮（按钮名与窗口标题相同）。当桌面上打开了多个窗口时，只需单击相应的按钮，就可以在多个窗口之间方便地进行切换。这也是任务栏最主要的功能之一。

任务栏右端凹下的矩形区域包含多个状态指示器。根据系统配置的不同，该区域中状态指示器的个数和内容也不同，如数字时钟、文字输入法状态及声音等。

(2)【开始】菜单

用鼠标左键单击任务栏左端的【开始】按钮，则弹出【开始】菜单，如图 5-3 所示。通过该菜单，用户可以快速启动其他应用程序、查找文件及获得帮助。

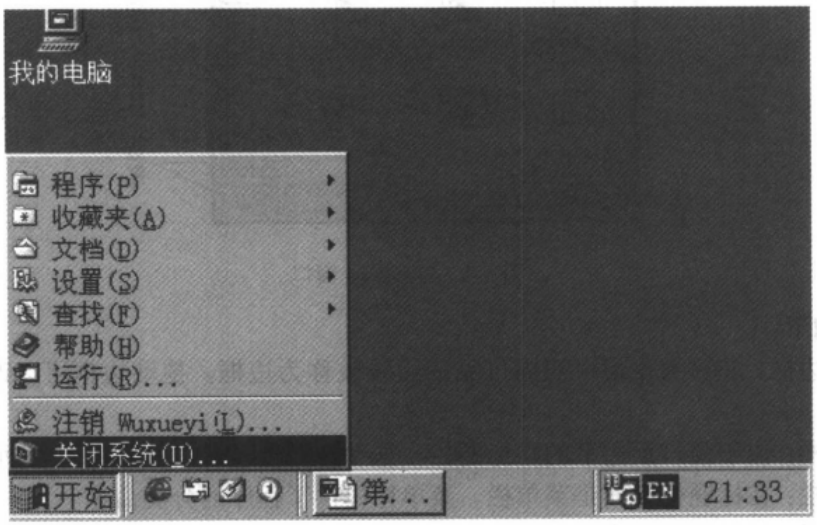


图 5-3　【开始】菜单

在【开始】菜单中，有的选项旁边带有三角形标记（如程序、文档等），将鼠标指向该选项时，系统会自动弹出下一级子菜单。这种子菜单被称为级联菜单。

操作提示：用户也可以通过键盘，按 Ctrl + Esc 键或按键盘上的窗口键来启动【开始】菜单。

启动应用程序的方法如下。

- 利用【程序】菜单启动应用程序：单击【开始】→【程序】→应用程序项；
- 利用【运行】命令启动应用程序：单击【开始】→【运行】，弹出【运行】对话框，输入路径和程序名，再单击【确定】按钮就可运行该程序。也可以单击【浏览】按钮，从下拉列表中选取某个程序来运行。

- 使用【文档】菜单启动应用程序：单击【开始】→【文档】→文档名称。

(3) 退出 Windows 98

使用 Windows 98 的用户，在关机前一定要正确退出 Windows 98；否则，可能会破坏一些未保存的文件和正在运行的程序。

退出 Windows 98 的正确方法：单击【开始】→【关闭计算机】→【关闭 Windows】。

3. 窗口的基本组成元素

窗口是 Windows 98 中最基本的元素。系统运行一个程序、打开一个文档或打开一个文件夹时，即可在屏幕上开辟一个矩形区域以显示相关信息。这个矩形区域就被称为窗口，如图 5-4 所示。

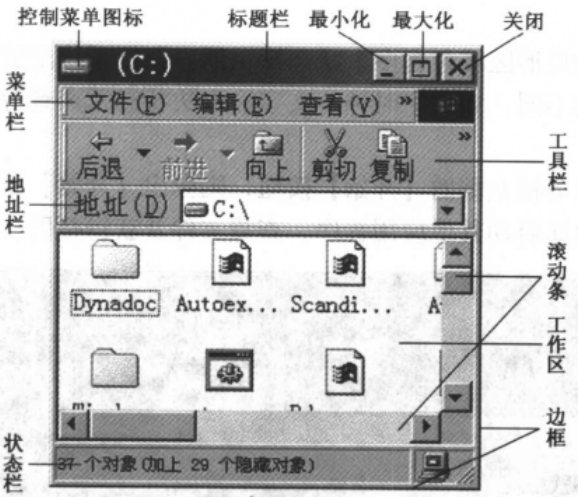


图 5-4 Windows 窗口

其组成如下。

- 窗口边框：非极大化窗口四周可见的边线被称为边框。拖动边框可以改变窗口的宽或高。
- 标题栏：位于窗口顶边下的是标题栏。拖动标题栏，可以在桌面上移动窗口。
- 菜单栏：在标题栏下面是菜单栏。菜单栏提供了程序的分类命令。
- 工具栏：包括了一些常用的功能按钮，如【文件复制】、【删除】等。
- 状态栏：在窗口底部，可显示窗口的状态。
- 最大化、最小化和关闭按钮：位于窗口右上角，用于改变窗口的大小或关闭窗口。
- 滚动条：包括水平滚动条和垂直滚动条，用于滚动显示窗口内容。
- 工作区：窗口内部的其他区域被称为工作区。

4. 对话框中常见的组成元素

Windows 98 系统为了完成某项任务而需要从用户那里得到更多的信息时，就显示一个对话框。顾名思义，对话框是系统与用户对话、交换信息的场所，是窗口界面的重要成分。

对话框中的标准部件包括文本框、列表框、单选按钮、复选按钮及命令按钮等。图 5-5 中给出了对话框中几种常见的组成部件。

(1) 文本框

文本框是用于输入文本信息的一个矩形区域。例如，图 5-5 中所示的【宽度】文本框用于输入一个宽度值。

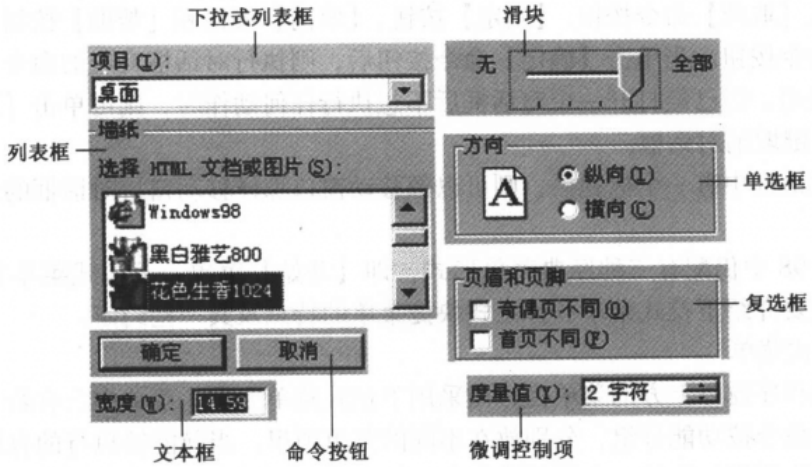


图 5-5　对话框中几种常用的组成部件

(2) 列表框

列表框是一个显示多个选项的小窗口。用户可以从中选择一项或几项。例如，在列出的文件清单中选择一个（或几个）要打开的文件。如果窗口尺寸容纳不下里面的内容，则窗口旁会有滚动条出现，可帮助你在列表中快速浏览和选择，如图 5-5 中所示的【墙纸】列表框。

(3) 下拉式列表框

下拉式列表框与列表框的不同点在于其初始状态是一个只包含当前选项（隐含选项）的小窗口。当单击窗口右边的下三角箭头按钮时，一个可供选择的列表窗口便会弹出。如图 5-5 所示的【项目】下拉列表框，单击下三角箭头按钮时，可供选择的各种屏幕保护程序便会出现。在多数情况下，不需要选择而采用隐含值的场合，采用下拉式列表框可以少占显示空间。

(4) 单选框

单选框是一组互相排斥的选项，任一时刻必须且只能从中选择一个。单击选项前面的“○”形按钮即可选中该项。被选中的状态为“⊙”，而原来被选中的项立刻变为未选中状态。如图 5-5 所示【方向】单选框。

(5) 复选框

复选框是一个可以开或关的任选项。单击选项前面的“□”形图标，则即可变成“☒”形，再单击一次，又回到未选中状态。在很多对话框中，常列出若干个复选框，用户可根据自己的需要来选取其中的一些选项。

(6) 微调控制项

微调控制项位于文本框右侧，是用于增减数值的一对箭头。如图 5-5 所示度量值文本框，其右侧就配有微调控制项。

(7) 滑块

滑块的工作方式很简单，向一个方向移动，值将增加；向另一个方向移动，值将减少，

如图 5-5 所示。

(8) 命令按钮

命令按钮上面都刻有自己的名字。单击命令按钮便可以立即启动一个动作。如图 5-5 所示的【确定】、【取消】命令按钮。【确定】按钮、【取消】按钮和【帮助】按钮是各对话框中几乎都有的命令按钮。当单击【确定】命令按钮后，则执行对话框对应的命令。该对话框也同时被正常关闭。当已经打开一个对话框后不想执行任何动作时，则可单击【取消】命令按钮或按下 Esc 键取消对话框。

如果在对话框中有一个标题栏，则可以像移动窗口那样移动窗口对话框的位置。

5. 菜单

Windows 98 中仍配有三种经典菜单形式，即【开始】菜单、下拉式菜单和弹出式快捷菜单。下面主要介绍下拉式菜单和弹出式快捷菜单的特点及其一些约定。

(1) 下拉式菜单

位于应用程序窗口上方的菜单栏，均采用下拉式菜单方式。菜单中含有若干条命令。为了便于使用，命令按功能分组，分别放在不同的菜单项里。当前能够执行的有效菜单命令以黑色显示，不能使用的无效命令则呈浅灰色。如果菜单命令旁带有“…”，则表示选择该命令后可弹出一个对话框，等待用户输入必要的信息或做进一步的选择。菜单项后面如果带有一个向右的三角形，则说明该菜单项下还有子菜单。如果菜单的后面带有一些符号，其含义是该菜单命令的快捷键。

(2) 弹出式快捷菜单

Windows 98 提供了一种随时随地可为用户服务的与上下文相关的弹出菜单。将鼠标指向某个已选中的对象或屏幕上的某个位置，单击鼠标右键，即可打开一个弹出式菜单。该菜单列出了与目前正在执行的操作直接相关的命令。例如，将鼠标指向一个文件夹窗口内部，单击鼠标右键，即会弹出如图 5-6 所示的快捷菜单。从中可以看出，菜单中的内容都是与文件夹及其窗口相关的命令。

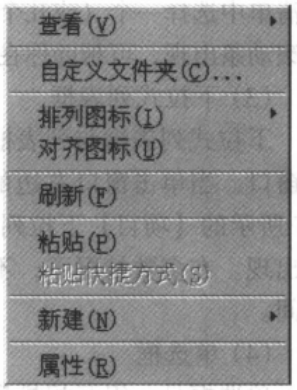


图 5-6 弹出式快捷菜单

快捷菜单中的这些命令是与上下文相关的，即根据单击鼠标右键时，鼠标指针所指的对象和位置的不同，弹出的菜单命令内容也不同。

5.1.3 Windows 98 文件管理

在 Windows 98 中，用户可以通过【我的电脑】窗口浏览文件、文件夹及其他一些系统资源，也可以通过【资源管理器】窗口浏览系统资源，从而完成各种操作，如复制、移动、删除、更名及查找等。

1.【我的电脑】窗口的操作

在 Windows 98 桌面上，用鼠标左键双击【我的电脑】图标，即可以打开【我的电脑】窗口。用户可以根据自己的需要，通过使用【查看】菜单的【工具栏】和【状态栏】命令，来决定是否显示工具栏和状态栏。

(1) 文件夹内容的显示方式

当打开一个文件夹后，文件夹中的内容就会显示在窗口中。使用【查看】菜单命令或工具栏上的【查看】图标，即可以改变文件夹中的内容显示方式。Windows 98 提供了四种显示方式：

- 以大图标显示：在这种显示方式中，对象的图标较大，适用于文件夹内容较少的情况；
- 以小图标显示：在这种显示方式中，对象的图标较小，适用于文件夹内容较多的情况；
- 列表显示：对象的图标是垂直排列的；
- 详细资料：文件窗口以列表方式显示详细内容，包括文件的名字、大小、类型及日期。

用户可根据具体情况选择一种合适的显示方式，同时还可以按对象名称、大小、日期项进行排序，以便于日后查找。

(2) 文件夹选项

选择【查看】菜单中的【文件夹选项】命令，即会弹出一个【文件夹选项】对话框，如图 5-7 所示。该对话框中有【常规】、【查看】及【文件类型】三个选项卡。

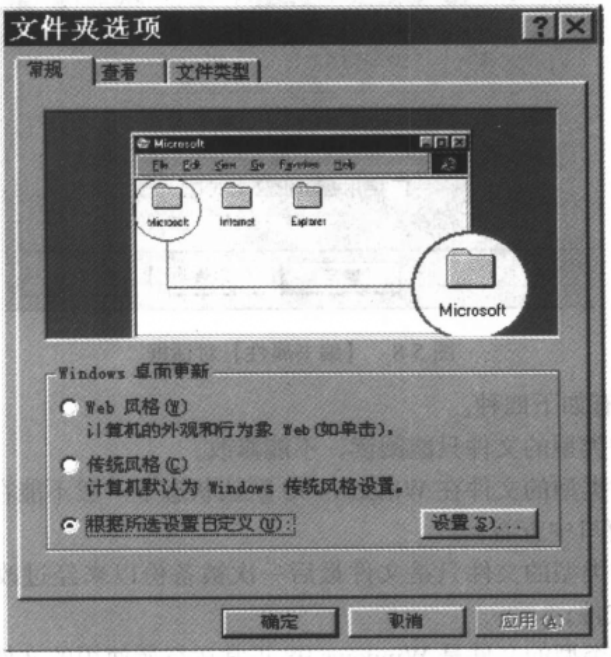


图 5-7　“文件夹选项”对话框

在【常规】选项卡中，用户可以选择窗口中文件系统显示的样式，如用 WEB 风格、使用 Windows 98 标准设置或根据选择设置。

在【查看】选项卡中，可以改变文件的显示方式，如在标题栏中显示完整路径、隐藏已知文件类型的文件扩展名、鼠标指向文件夹和菜单项时显示提示信息。

在【文件类型】选项卡中，可以进行文件类型的注册。某种文件类型被注册后，Windows 98 会自动识别该类型的文件。当双击该类型文件时会自动利用与文件相关联的应

用程序打开该文件。

利用【文件类型】选项卡，用户可以向系统中添加新的文件类型，也可以删除某类型文件在系统中的注册。

(3) 文件夹和文件属性

在此对话框中，用户可以查看该文件夹或该文件的相关信息。

用户要查看或修改文件的属性，可从【文件】菜单中选择【属性】命令，或在该文件上单击鼠标右键，从弹出的快捷菜单中的选择【属性】命令，即可打开该文件的【属性】对话框，例如图 5-8 所示的【编书属性】对话框。

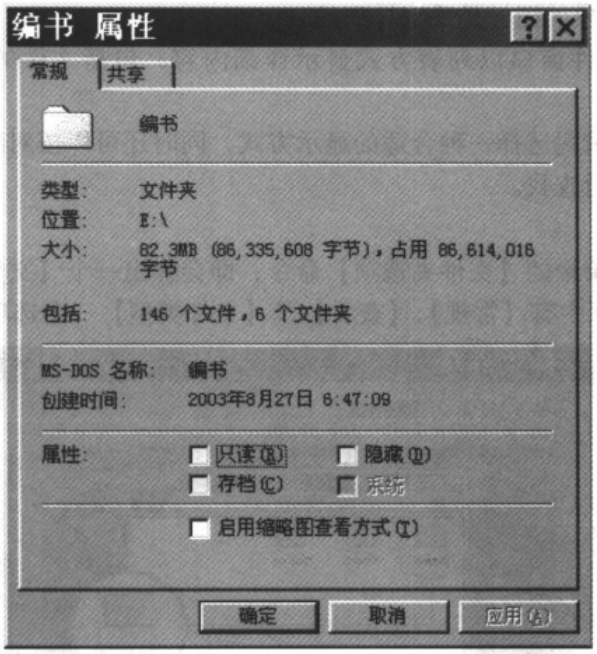


图 5-8 【编书属性】对话框

文件的属性主要有如下四种。

- 【只读】：这种类型的文件只能阅读，不能修改；
- 【隐藏】：这种类型的文件在 Windows 98 默认状态下一般不能在【我的电脑】和【资源管理器】窗口中查看；
- 【存档】：这种类型的文件只是文件最后一次被备份以来经过改动的文件，普通文件基本是此类文件；
- 【系统】：这种类型的文件是 Windows 98 正常运行所使用的文件。在默认情况下，它们不显示在文件列表中。

2. 资源管理器窗口的操作

资源管理器是 Windows 98 提供的一个查看和管理计算机上所有资源的应用程序，通过它可以运行程序、打开文档、将文件夹从一个地方移动或复制到另一个地方、格式化和复制软盘，甚至可以浏览 WEB 网页。下面就介绍资源管理器的使用。

(1) Windows 98 资源管理器窗口的布局

Windows 98【资源管理器】采用双窗口显示结构，如图 5-9 所示。左窗口（所有文件夹

窗口）显示整个系统的文件夹树。它将系统中所有的资源用一个树状结构的框架显示出来。文件夹树更形象地描述了磁盘文件中上下层次分明的组织结构，包括桌面、我的电脑、驱动器和文件夹等不同的图标。右窗口显示的是左窗口中被选中对象的具体内容。

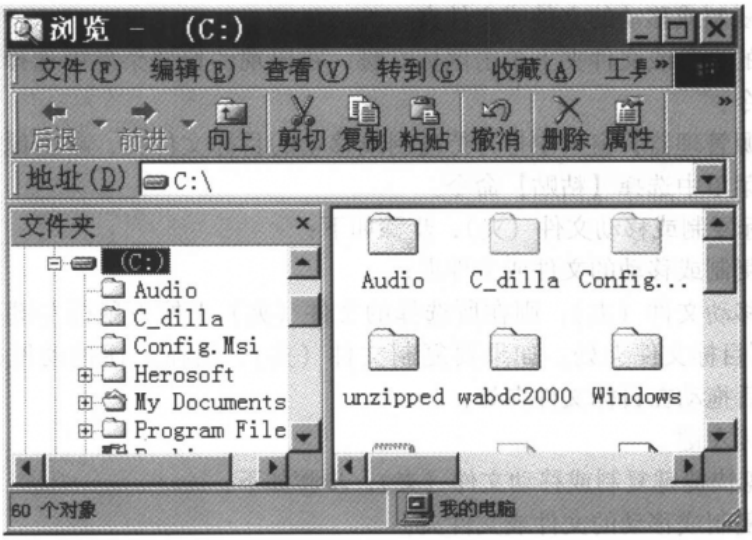


图 5-9　【资源管理器】窗口

在左窗口中，有些文件夹图标左边有一个方框。如果方框中是【+】号，则表示该文件夹中还包含有文件夹。这时，如用鼠标左键单击【+】号或双击此文件夹图标，则【+】号即可变为【-】号，同时在下面列出内层文件夹；再用鼠标左键双击文件夹或单击【-】号，则【-】号又变为【+】号，且内层文件夹将隐藏起来。

若用鼠标左键双击右窗口中的文件夹图标，则窗口中变为该文件夹中的内容。

若将鼠标指针置于左、右窗口分界处，则指针形状会变成左右双向箭头。此时按下鼠标左键拖动分界线，即可改变分界位置。

(2) 文件或文件夹的复制和移动

复制和移动文件（夹）是文件管理中最常用的操作。用户可以把文件或文件夹复制、移动到其他磁盘驱动器中。

文件（夹）的复制或移动可以通过以下几种方法来实现。

① 使用菜单命令复制或移动文件（夹）。在【资源管理器】中，使用菜单命令复制或移动文件（夹）的步骤如下：

- 选定待复制或移动的文件或文件夹；
- 选择【编辑】菜单中的【复制】或【剪切】命令；
- 在【资源管理器】窗口中打开要复制或移动后的目标文件夹；
- 选择【编辑】菜单中的【粘贴】命令，被复制或移动的文件或文件夹就会出现在目标文件夹中。

② 用工具栏复制或移动文件（夹），步骤如下：

- 选定待复制或移动的文件或文件夹；
- 单击工具栏中的【复制】或【剪切】按钮；

- 在【资源管理器】窗口中打开待复制或移动的目标文件夹，单击工具栏上的【粘贴】按钮。

③ 使用快捷菜单复制或移动文件（夹），步骤如下：

- 选定待复制或移动的文件或文件夹；
- 在选择的文件或文件夹上单击鼠标右键，在出现的快捷菜单中选择【复制】或【剪切】命令；
- 在【资源管理器】窗口中打开待复制或移动的目标文件夹，单击鼠标右键，在出现的快捷菜单中选择【粘贴】命令。

④ 使用鼠标复制或移动文件（夹），步骤如下：

- 选定待复制或移动的文件或文件夹；
- 如果要移动文件（夹），则在所选择的文件（夹）上按下鼠标左键不放，拖动文件（夹）到目标文件夹处。如果要复制文件（夹），则在上述拖动鼠标的同时按 Ctrl 键，然后拖动在目标文件夹处；
- 松开鼠标左键。

⑤ 使用键盘快捷键复制或移动文件（夹），步骤如下：

- 选定待复制或移动的文件或文件夹；
- 如果要移动文件（夹），则按 Ctrl＋X 组合键；如果要复制文件（夹），按 Ctrl＋C 组合键；
- 在【资源管理器】窗口中打开目标文件夹；
- 按 Ctrl＋V 组合键。

(3) 文件（夹）的删除

如果用户不再需要某个文件或文件夹，则可以将它删除。但需要注意的是，当用户删除一个文件夹时，该文件夹中所有文件和子文件夹都将被删除，因此在执行删除操作时，应当确认是否要删除文件夹中的全部内容。

① 使用菜单命令删除文件（夹），步骤如下：

- 选择待删除的文件（夹）；
- 从【文件】菜单中选择【删除】命令，此时会弹出如图 5-10 所示的【确认文件夹删除】对话框，提醒用户是否真的希望删除该文件（夹）；

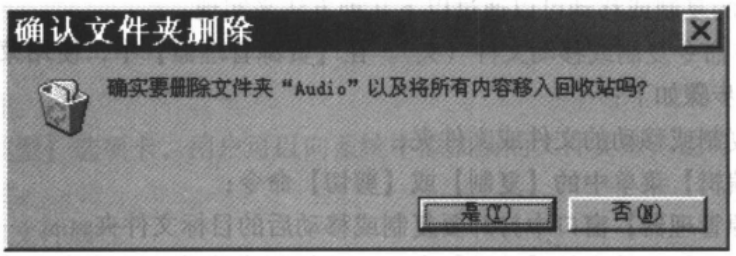


图 5-10 【确认文件夹删除】对话框

- 根据需要，单击【是】或【否】按钮。

② 用工具栏删除文件（夹），步骤如下：

- 选择待删除的文件（夹）；

↘ 单击工具栏上的【删除】按钮。

③ 使用快捷菜单删除文件（夹）的步骤如下：

↘ 选择待删除的文件（夹）；

↘ 单击鼠标右键，弹出快捷菜单，单击【删除】命令。

④ 使用鼠标拖动删除文件（夹）的步骤如下：

↘ 选择待删除的文件（夹）；

↘ 将鼠标指针移到该文件（夹）上并将其拖到回收站。

(4) 文件或文件夹的更名

在【资源管理器】窗口中，用户可以直接更改文件或文件夹的名称。其方法有如下几种：

↘ 选择某个文件（夹）后，在【文件】菜单中单击【重命名】命令，此时用户就可在该文件（夹）的名称框内输入新的文件（夹）名称。

↘ 在某个文件（夹）上单击鼠标右键，即可从弹出的快捷菜单中单击【重命名】命令，从而更改该文件（夹）的名称。

↘ 选择某个文件（夹）后，在该文件（夹）的名称上单击，即可输入新的文件（夹）的名称。

(5) 创建新的文件（夹）

在【资源管理器】窗口中，可以创建新的文件夹、新的对象及快捷方式。

① 创建新的文件夹的步骤如下：

↘ 选定创建新文件夹的上级文件夹；

↘ 在文件菜单中单击【新建】，在其子菜单中单击【新文件夹】命令；

↘ 在高亮的新文件夹框中输入新文件夹名称。

② 创建新文档。当用户要创建一个新文档，如文本文件、图像文件等时，应先在文件菜单中单击【新建】，再在其子菜单中选择新建文档的类型，这时该类型文档所关联的应用程序将被启动，即可创建新的文档。

③ 创建快捷方式，即创建快速打开文件或文件夹、应用程序等。快捷方式的图标可以放在桌面上，也可以放任意文件夹中。其创建步骤如下：

↘ 选定放置【快捷方式】的文件夹；

↘ 在文件菜单中用鼠标左键单击【新建】命令，在其子菜单中用鼠标左键单击【创建快捷方式】命令，随后按快捷方式向导提示操作即可完成。

(6) 查找文件（夹）

利用查找程序即可以迅速定位文件所在的位置。用户可以利用【查找】对话框中的选项卡，定义更详细的搜索条件，以快速找到所用的文件。

查找文件（夹）主要有以下三种方式：

↘ 使用【资源管理器】中工具子菜单中的【查找】命令进行查找；

↘ 使用【我的电脑】中【文件】子菜单中的【查找】命令进行查找；

↘ 使用【开始】菜单中的【查找】命令进行查找。

(7) 格式化软盘

格式化软盘是操作系统的一个常用功能。在 Windows 98 下，磁盘格式化的操作步骤如下：

- 把要格式化的软盘插入驱动器中；
- 在【资源管理器】中选定该驱动器图标，选择【文件】菜单中的【格式化】命令，即可弹出如图 5-11 所示对话框，选择格式化方式、容量及一些选项后，单击【开始】按钮即可完成。

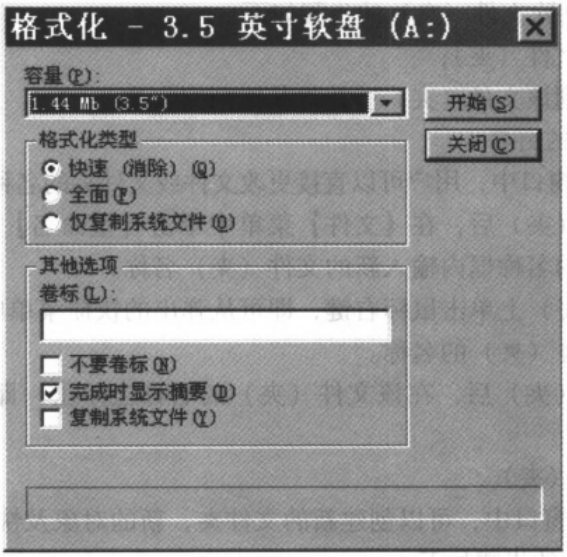


图 5-11 【格式化－3.5 英寸软盘（A:）】对话框

5.2 中文版 Word 操作

Microsoft 公司推出的中文版 Office 软件，是基于中文 Windows 9x 工作平台的。Office 中文版组成部分之一就是 Word。它是功能强大的文字处理软件，具有录入、编辑、排版、预览及打印功能。Microsoft 公司自从 1991 年推出 Word 2.0 版本文字处理软件以来，Word 就在全世界范围内广泛流行开来。随后推出的 Word 5.0、Word 6.0 版本的文字处理软件更为广大计算机用户所喜爱。Word 具有强大的文字处理和排版功能。一般的用户利用它即可以高效率、高质量地书写及处理书信、备忘录、商务计划、科技论文、研究报告等。Word 以其“所见既所得”的显示方式、“软件易用”、“功能齐全”而得到广泛的应用。

5.2.1 中文版 Word 简介

1. Word 的安装

Microsoft 公司提供的 Office 系列软件有软盘版和光盘版。因此，Word 的安装有从软盘安装和从光盘安装两种方法。

由于从光盘安装方便快捷，故下面将介绍从光盘安装 Office 中文版 Word 的方法和步骤：

- 启动 Windows 9x，进入 Windows 9x 桌面；
- 插入 Office 中文版安装光盘；
- 打开【资源管理器】，找到光盘目录中的 Setup.exe 安装程序后，用鼠标左键双

击它；

- 安装程序开始后，要求插入光盘所附带的安装许可磁盘，确认是合法用户后，即可开始安装；
- 按照屏幕提示，输入用户名、公司名等。

遵循安装程序的提示，即可完成安装。

2. 中文 Word 的启动

中文 Word 的启动有如下几种方法。

- 利用菜单启动 Word：用鼠标左键单击【开始】→【程序】→【Microsoft Word】。
- 在桌面上制作一个 Word 快捷键（可执行文件名为 Winword.exe），只要用鼠标左键双击此快捷键，就可进入 Word 主窗口。
- 利用 Microsoft Office 的快捷工具栏。每当打开计算机时，在屏幕的右上方即可显示 Office 的快捷工具栏，其中列有应用程序的快捷图标。用鼠标左键单击 Word 的快捷图标，也可快速进入 Word 主窗口。

3. 中文 Word 的工作界面

中文 Word 的工作界面由文本编辑区和各种工具栏组成，如图 5-12 所示。

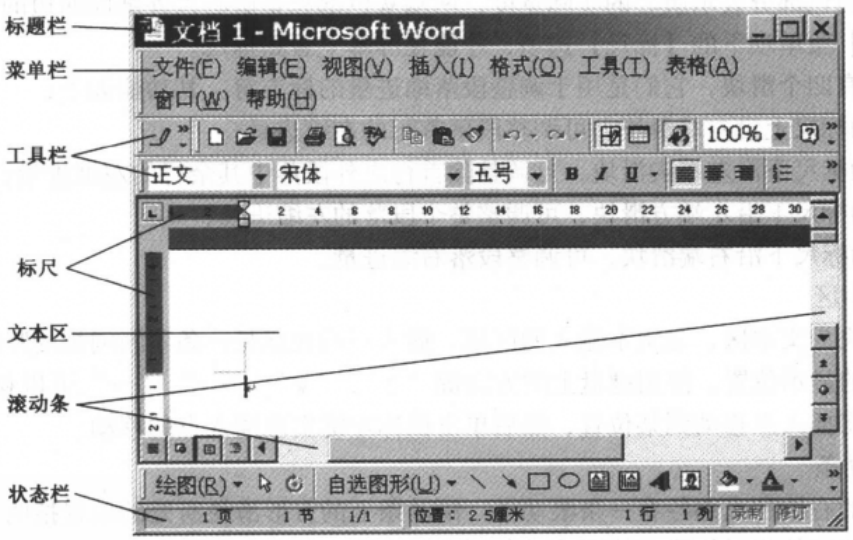


图 5-12　中文 Word 工作界面

(1) 标题栏

Word 窗口最上方的一条为标题栏。其中，“Microsoft Word”是中文 Word 的名称。其前面是 Word 当前处理的文档名称。进入 Word 后，Word 会自动建立一个新的文档，并起名为“文档 1”，以后再新建立的文档则依次起名为“文档 2”、“文档 3”等，据此来标记那些未曾存盘命名的文档。如果调入的是存在盘上的文档，则该处显示该文档的实际名称。

(2) 菜单栏

标题栏下面的一行是菜单栏，包括九项菜单。它们分别是【文件】、【编辑】、【视图】、【插入】、【格式】、【工具】、【表格】、【窗口】和【帮助】。这些菜单都是下拉式的。打开它们有两种方法：一种方法是使用鼠标，移动鼠标指针并指向所要打开的菜单名上，单击鼠标左

键，菜单立即被打开；另一种方法是使用键盘，先按 Alt 键激活菜单栏，再按菜单名后的字母即可打开菜单。打开菜单后可见，菜单内的选项实际上是一组命令的集合。用鼠标左键单击某命令后即可执行该命令。命令项均带有一个由括号括起来的字母，表示按该字母键后即可执行相应的命令。有些命令项后还带有快捷键，如【新建】命令后的【Ctrl＋N】。其含义是在编辑状态下按 Ctrl 键不放，再按 N 键就可以直接执行【新建】命令。显然，从菜单中选取执行命令需进行两次操作，而使用快捷键则只需要一次操作，故使用快捷键效率更高。

(3) 工具栏

菜单栏下面是工具栏。工具栏上排列着一系列直观形象的按钮，分别对应于特定的常用操作命令。用鼠标左键单击工具栏上的按钮，即可执行相应的操作命令。工具栏的使用，使文档编辑工作变得轻松而易学。用户经常使用的是常用工具栏和格式工具栏。常用工具栏可完成文件的建立、存储、预览及打印等工作；格式工具栏可完成文件的编辑、排版等工作。中文 Word 提供了十几种工具栏，可以在需要的时候加载到屏幕上。具体的加载方法是：用鼠标左键单击【视图】菜单中【工具栏】命令，再用鼠标左键单击所需工具栏。

(4) 标尺

工具栏的下方是标尺，上面有刻度和数字。用户可以借助标尺精确定位，设置页边距。标尺中部的白色部分表示版心的实际宽度，两端灰色部分是版心以外纸面四边的空白。只有选中【视图】菜单项下的【标尺】选项，才能在屏幕上看到标尺。

标尺上有四个滑块，它们是用于调整段落缩进量的标志符。其功能如下：

- 移动标尺上沿左端滑块，可调整段落首行的左缩进量；
- 移动标尺下沿左端尖滑块，可调整除首行之外段落中其余行的左缩进量；
- 移动标尺下沿左端方滑块，可调整整个段落的左缩进量；
- 移动标尺下沿右端滑块，可调整段落右缩进量。

(5) 文本区

屏幕中央为文本区，是文本输入的区域。插入点的光标是一条不断闪烁的竖线，它标记新键入字符的显示位置。使用键盘上的方向键“↑”、“↓”、“←”、“→”可以移动插入点，也可用鼠标将插入点移动到某位置，然后单击鼠标左键实现插入点的移动。

(6) 滚动条

文本区的右侧和底部各有一条滚动条。滚动条上的方形滑块所处的位置指明了当前插入点在整个文档中的相对位置。使用滚动条可以迅速地查阅文本区的内容。

- 单击垂直滚动条两端的三角形按钮，可使文本区中的内容向下或向上滚动一行。
- 单击水平滚动条两端的三角形按钮，可使文本区中的内容向右或向左移动一列。
- 拖动滚动条上的方形滑块，可使文本区内容随意移动。

另外，使用 PageUp 和 PageDown 键也可以实现文本的整屏滚动。

(7) 状态栏

Word 窗口最下面的一长条为状态栏，用于显示当前编辑窗口信息或某些操作的简明信息。在编辑状态下，状态栏上各项含义如下：

- 页、节——显示插入点所在页、所在节；
- 1/10——当前所在页/当前文档总页数；
- 行、列——插入点在当前页中的行号和列号；

- 修订——以黑字显示时表明修订标志被激活；
- 改写——以黑字显示时表明文字录入处于改写模式，可在改写和插入模式间进行切换。

5.2.2　文档的基本编辑

1. 建立新文档

建立一个新文档，需进行如下操作步骤：

- 单击【文件】菜单；
- 选择【新建】命令，打开如图 5-13 所示的【新建】对话框。

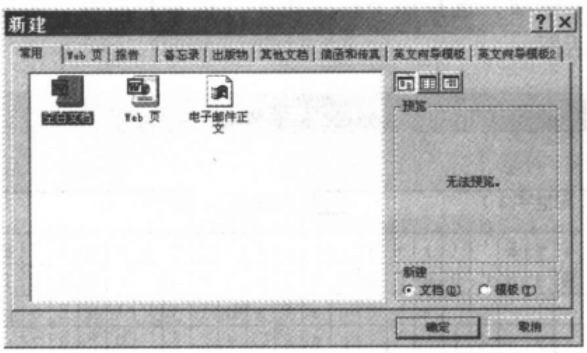


图 5-13　【新建】对话框

2. 输入文档内容

建立了新文档以后，就可以输入文档内容了。文档编辑窗口中有一个闪烁的棒状光标，当鼠标移到文档窗口中时会为成“I”形状。闪烁光标的位置就是文档输入的位置。若输入英文、中文字符时，字符就显示在闪烁光标所在的位置上。Word 在此输入过程中可设置自动更正单词和一些汉语成语等功能。

(1) 中、西文切换

在安装 Windows 9x 中文版操作系统时，已经预先设置了全拼、双拼、郑码、表形码、区位及智能 ABC 等汉字输入法。若想进入汉字输入状态，则可用鼠标左键单击任务栏右下端输入提示区的【En】按钮，系统便会弹出一个菜单。用鼠标左键单击某一个菜单项，屏幕的输入方式便切换到相应的方式之下。例如，用鼠标左键单击【智能 ABC】菜单项，屏幕上便出现一个该方式状态的窗口，此时键盘上的输入方式即为【智能 ABC】汉字输入方式。若要切换为西文输入方式，则只要在任务栏上用鼠标左键单击刚才的按钮处，再用鼠标左键单击【En】按钮，就可转换为英语（字符）输入方式。

用 Ctrl + Space 键来启动或关闭选用的中文输入法。使用 Ctrl + Shift 键或 Alt + Shift 键在英文和各种输入法之间也可进行转换。进入汉字输入状态后，可出现如图 5-14 所示按钮。

(2) 全角/半角切换

如图 5-14 所示中的第三个按钮是进行【全角/半角切换】按钮。用鼠标左键单击此按钮即可进行中文输入的全角/半角间切换。全角状态时，月牙形变成实心圆形。



图 5-14　汉字输入按钮

用 Shift + Space 键也可以完成上述切换功能。

(3) 中、英文标点切换

如图 5-14 所示中的第四个按钮是【中/英文标点符号切换】按钮。在中文输入状态时，用鼠标左键单击此按钮后，再输入的符号将变成中文标点符号；反之亦然。

(4) 输入一段文本的基本操作

在 Word 普通视图下的文档工作窗口中输入一段文本。其输入过程中若出现错误，则可按“BackSpace”键将其删除，再重新输入。在一段文本结束之前不要按Enter 键，当一行文本满行之后，Word 会自动换行。

若要输入符号及特殊字符，则其操作如下。

- 单击【插入】菜单项，选择该下拉菜单中【符号】命令，屏幕会显示如图 5-15 所示的【符号】对话框。

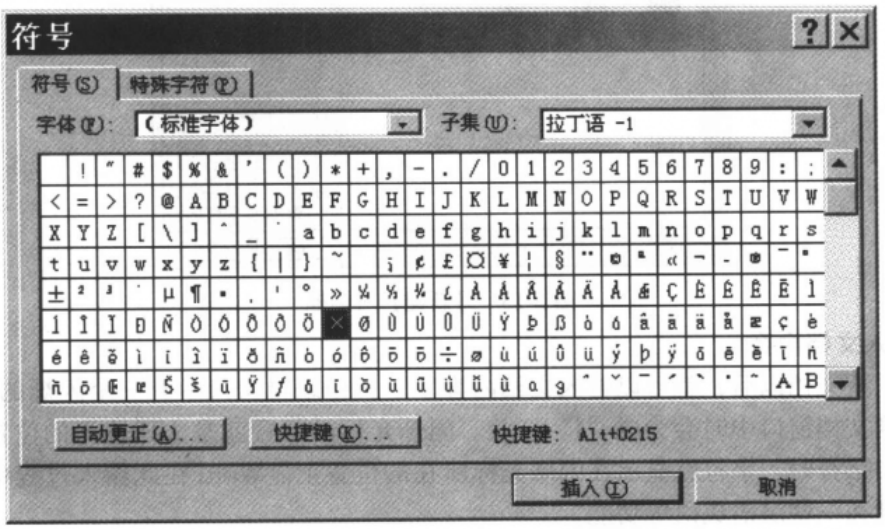


图 5-15 【符号】对话框

- 单击【符号】对话框中的【符号】或【特殊字符】选项卡，选中某符号后，按【插入】按钮。

3. 保存文档

文档的输入结束后，要进行保存文档的操作。

(1) 利用菜单保存文件

- 单击【文件】菜单，选择【保存】命令，弹出如图 5-16 所示的【另存为】对话框。
- 【保存位置】列表框用来选择或查找欲保存的文件夹或驱动器。
- 【文件名】列表框用来输入该文档的名字。若进行保存文档操作时，则一般将以默认的文件名保存。
- 【文档保存类型】列表框用来选择该文档的保存格式。Word 中文版提供有 Word 文档、纯文本及文档模板等多种格式。
- 单击【保存】按钮，完成保存操作。如果要取消本次操作，单击【取消】按钮。

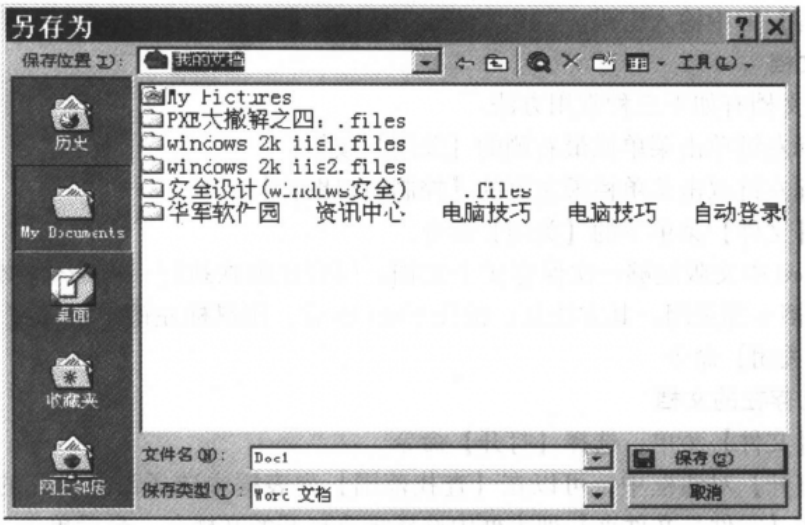


图 5-16　【另存为】对话框

(2) 建立自动保存文档功能的操作步骤如下。

- 单击【工具】菜单，选择【选项】命令，并在随之出现的对话框中选择【保存】选项卡，如图 5-17 所示。

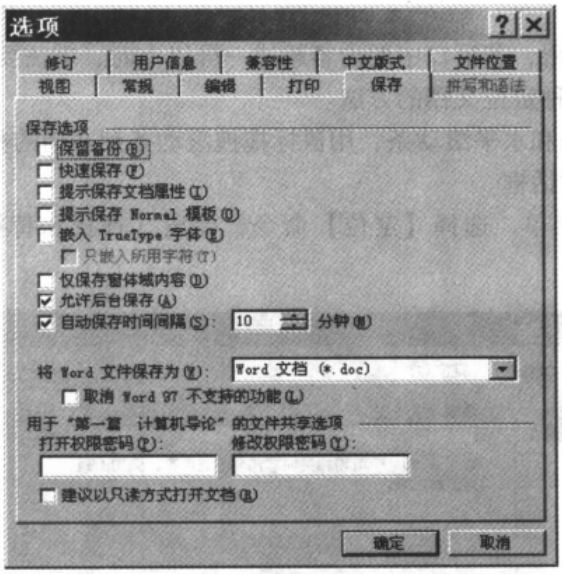


图 5-17　【保存】选项卡

- 选定【自动保存时间间隔】复选框，即在复选框前的小方格中，用鼠标左键单击此位置，就会出现一个对号。
- 在【分钟】微调框中输入或调整时间。
- 用鼠标左键单击【确定】按钮。
- 若用户不希望其他人随意查看或修改其文档，则可在【打开权限密码】或【修改权

限密码】框中输入密码。

4. 关闭文档

关闭一个文档有如下三种常用方法。

- 用鼠标左键单击菜单栏最右侧的【关闭】按钮；
- 用鼠标左键双击菜单栏最左侧的【控制窗口图标】；
- 执行【文件】菜单下的【关闭】命令。

说明：Word 中文版能够一次保存多个文档，同样也能在执行一次命令的情况下将同时编辑的多个文档全部关闭。其方法是：按住 Shift 键后，用鼠标左键单击【文件】菜单，然后单击【全部关闭】命令。

5. 打开已存在的文档

- 单击【文件】菜单，选择【打开】命令。
- 在【打开】对话框中，可以在【查找范围】列表框中选择需要打开的文档的位置，然后在【文件与文件夹】列表框中选择需要打开的文档名，最后单击【打开】按钮。也可以直接在【文件名】文本框中直接输入所需打开文档的完整路径和文档名，最后用鼠标左键单击【打开】按钮。

6. 文本的编辑

在 Word 中文版中，文字的基本编辑方法有文档的定位、文本的选择、编辑、复制、粘贴及文字的查找、替换等。

(1) 定位文档

文档窗口的大小有限，若窗口中的内容超过一屏以上时，就需要借助于定位文档功能来查阅文档。下面介绍几种定位文档的方法。

① 利用垂直滚动条和水平滚动条，用鼠标拖拽滚动条到文档的相应位置。

② 利用【定位】对话框。

- 单击【编辑】菜单，选择【定位】命令，弹出【查找和替换】对话框，如图 5-18 所示。

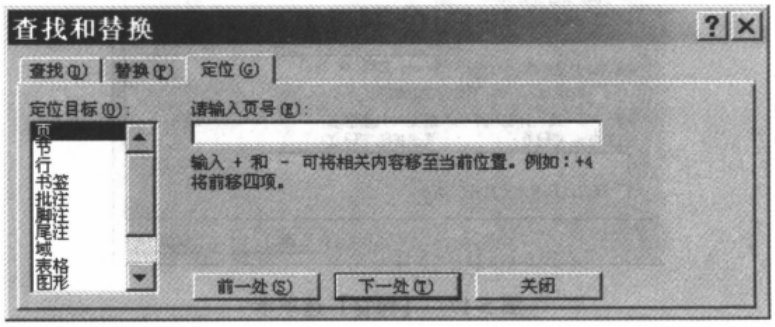


图 5-18 【查找和替换】对话框

- 在对话框的【定位】选项卡中设定需要定位的目标。例如，在【定位目标】列表中选择【页】，在【请输入页号】文本框中输入页号，如选 3，这样就可以将光标定位到文档的第 3 页，Word 将显示第 3 页的文档内容。当然也可以选择其他的定位目标，如行、节、批注等。

③ 建立书签浏览。书签是用来在书本上做标记的，Word 文档中也能建立书签来标记所选文字、图形、表格或其他项目。

- 用鼠标左键单击【插入】菜单，选定其中的【书签】命令，就可显示如图 5-19 所示的对话框。
- 在对话框中，在【书签名】文本框中输入书签的名字，再用鼠标左键单击【添加】按钮，Word 自动关闭该对话框。
- 重新打开【书签】对话框后，创建的书签将显示在书签列表框中，在此列表中用户可以选择要定位的目标。

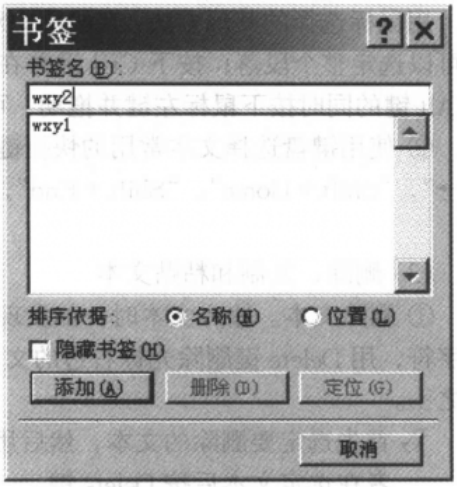


图 5-19　【书签】对话框

实质上，书签是文档定位的一种形式。在建立这种书签之前，必须先定位光标所在的位置，或者先选定要建立书签的图片、文字或表格等。

④ 使用键盘按键定位文档。使用键盘按键定位文档，有快速、准确的优点。下面是一些常用的定位文档的按键，如 Home，End，PageUp，PageDown，↑，↓，←，→，Ctrl + Home，Ctrl + End 键。

(2) 选择文本

若需对文档中的内容进行编辑和格式排版时，则应要先将相应的文本选定。选择文本可用鼠标选择或键盘选择。

① 使用鼠标选择文本，操作方法如下。

首先将“I”形鼠标指针移动到选定的字符的左边，再按下鼠标左键并同时拖动鼠标的光标，直到选定的字符右边，然后释放鼠标左键。此时，被选定的文档文本可以是一个英文字符或一个汉字，以至可以是文本的一段或整个文档。图 5-20 中反色显示部分被选中。

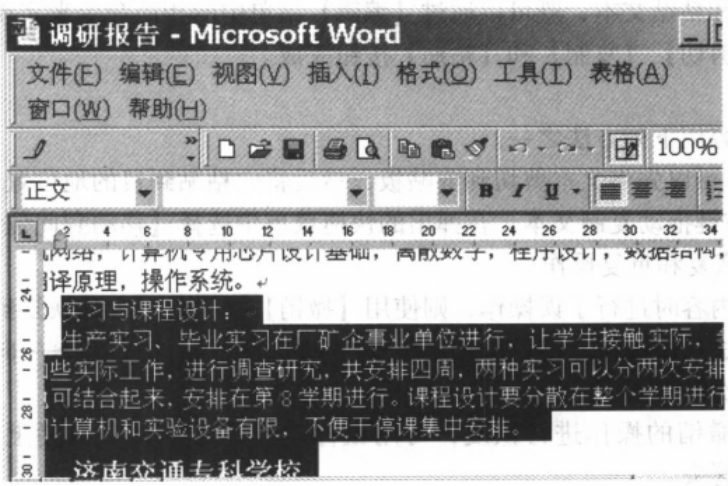


图 5-20　选择文本

用鼠标选择的其他方法：在段落中双击鼠标左键可以选定一个单词或分句；在段落中双击可以选定整个段落；按下 Ctrl 键并在任意句中用鼠标左键单击，可以选中一个句子；按下 Alt 键的同时按下鼠标左键并拖动，可以选定一整列。

② 使用键盘选择文本常用的快捷键有“Shift + ↑”，“Shift + ↓”，“Shift + ←”，“Shift + →”，“Shift + Home”，“Shift + End”，“Shift + PageUp”，“Shift + PageDown”，“Ctrl + A”等。

(3) 删除、复制和粘贴文本

① 删除文本。输入文本时，如发现了错误，就可以用 Backspace 键来删除光标左侧的文本字符，用 Delete 键删除光标右边的文本字符。如要删除大段的文本内容，则可采用下列方法之一。

- 首先选定要删除的文本，然后用鼠标左键单击【编辑】菜单中的【清除】命令，或者在选定文本后按 Delete 键。
- 选定要删除的文本后，然后用鼠标左键单击【编辑】菜单中的【剪切】命令。
- 在选定要删除的文本区内，单击鼠标右键打开快捷菜单，从菜单中选择【剪切】命令。

② 复制文本内容用下列方法之一。

- 选定要复制的文本，然后在菜单栏中选择【编辑】，在其下拉菜单中选择【复制】命令，移动光标到粘贴文本的目的地，执行【编辑】栏中的【粘贴】命令。
- 在选定的文本中按下鼠标右键并拖动至要放置文本的位置处再松开鼠标右键，此时会弹出一个快捷菜单，用鼠标左键单击快捷菜单中的【复制到此位置】命令，就可以将选定的文本复制到指定的位置。

③ 粘贴操作就是将剪贴板中的内容放到目的地的操作。

执行了复制或剪切命令后，文本就保留在剪贴板中。执行粘贴命令后，就是将剪贴板中的文本粘贴到目的地。执行粘贴命令后，在剪贴板中的内容不会消失，可以进行多次粘贴操作。

剪切、复制和粘贴文本，既可以通过【编辑】菜单中的相应命令来实现，也可以通过常用工具栏上的【剪切】、【复制】和【粘贴】按钮完成。

(4) 移动

移动文本内容用下列方法之一。

- 移动文本可以先将文本剪切到剪贴板，然后将它粘贴到目的地位置。
- 用鼠标右键拖动复制文本，在弹出的快捷菜单中选择【移动到此位置】命令项。

(5) 撤销、恢复和重复操作

如果在编辑内容时进行了误操作，则使用【撤销】操作就可自动恢复到上一次误操作之前的状态。其方法是用鼠标左键单击常用工具栏上的【撤销】工具钮——向左侧的箭头，可立即撤销最后一次操作。

恢复是对被撤销的操作进行恢复，可用鼠标左键单击常用工具栏上的【恢复】工具钮——向右侧的箭头。

重复操作是对上一次操作进行重复，按 F4 键或者按 Ctrl + Y 键进行重复操作。

(6) 查找和替换

在 Word 中进行文本编辑，可以搜索和替换文字，也可以搜索和替换单词的各种形式。例如，用“build”替换“make”，用“built”替换“made”。

① 查找的操作方法如下：

- 用鼠标左键单击【编辑】菜单的【查找】命令，在【查找内容】框内键入要搜索的文字；
- 若想继续查找，则用鼠标左键单击【查找下一个】按钮；若想结束查找，则按 Esc 键可取消正在进行的搜索。

② 替换的操作方法如下：

- 用鼠标左键单击【编辑】菜单的【替换】命令，出现如图 5-21 所示对话框；

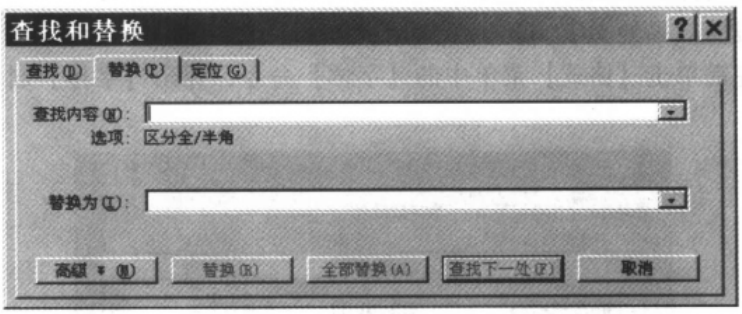


图 5-21　【查找和替换】对话框

- 在【查找内容】框内输入要搜索的文本，在【替换为】框内输入替换文字；
- 用鼠标左键单击【查找下一处】、【替换】或者【全部替换】按钮，或按 Esc 键取消正在进行的搜索。

(7) 插入公式

Word 中文版利用公式编辑器可以方便地在文档中输入和编辑自己需要的公式，而且所显示的公式样式就是打印成文后的式样。

- 在要插入公式的地方单击鼠标左键。
- 用鼠标左键单击【插入】菜单中的【对象】命令，然后用鼠标左键单击【新建】选项卡。
- 用鼠标左键单击【对象类型】框中的【Microsoft Equation3.0】选项。
- 用鼠标左键单击【确定】按钮。
- 通过从【公式】工具栏上选择符号及键入变量和数字的方式构造公式。在【公式】工具栏的上面一行，可以选择其中的 150 多个数学符号；在下面一行中，可以在包含如分式、积分、求和等符号的许多模板或框架中进行选择，如图 5-22 所示。

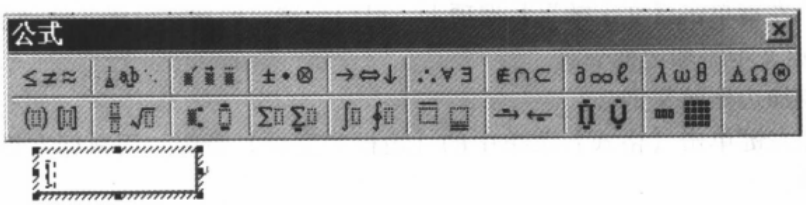


图 5-22　【公式】工具栏

- 完成公式的编辑后，在公式工具栏外单击鼠标，就可以退出公式编辑器，返回 Word 文档。

5.2.3 文档的格式化

录入文字后，需要对文字进行修饰和版面的格式化。格式化的内容丰富多彩，富于变化，是体现个性的地方。最基本的格式化操作包括设置字体、改变型号及设置页面等。

1. 设置字符格式

要为某一单词设置字符格式，则可用鼠标左键单击该单词。要为若干个单词或一个单词内的若干字母设置字符格式，则可选定此段文字。

(1) 设置文字的特殊效果

- 选中要设置效果的文字。
- 用鼠标左键单击【格式】菜单中的【字体】命令，弹出【字体】对话框，如图 5-23 所示。



图 5-23 【字体】对话框

- 用鼠标左键单击【字体】选项卡，可以设置字体、字号、字形等，还可以加一些特殊的效果。
- 用鼠标左键单击【字符间距】选项卡，可以设置字符间的距离及字符的位置。
- 用鼠标左键单击【文字效果】选项卡，可以为选定的文字设置各种动态效果。

(2) 为选定文本添加边框

- 选定要加边框的文字。
- 用鼠标左键单击【格式】菜单中的【边框和底纹】命令，用鼠标左键单击【边框】选项卡。
- 在【设置】栏中选择边框类型、宽度及颜色。

(3) 为选定文字添加底纹

- 选定要加底纹的文字。
- 用鼠标左键单击【格式】菜单中的【边框和底纹】命令，用鼠标左键单击【底纹】选项卡。
- 在【填充】栏中选择底纹的颜色，在【图案】栏中选择底纹的式样。

(4) 设置文字排版方向

用鼠标左键单击【格式】菜单中的【文字方向】命令，选择合适的排版方向。

2. 设置页面格式

页面的格式设置包括选择纸张大小、页面方向、页边距、页眉和页脚、页号和行号等。

(1) 选择纸张大小

用鼠标左键单击【文件】菜单中的【页面设置】命令，打开【页面设置】对话框，然后用鼠标左键单击【纸型】选项卡，选择合适的纸张型号；用鼠标左键单击【方向】下面的【纵向】或【横向】按钮，可以改变纸张的方向。

(2) 改变页边距

设置纸张的上、下、左、右的页边距，以使文字显示在纸张合适的位置。有两种方法可以改变页边距。

① 在文本区域直接改变，方法如下：

- 切换至页面视图或打印预览状态；
- 如果要改变左、右页边距，则可用鼠标指向水平标尺上的页边距边界，待鼠标箭头变成双向箭头后拖动页边距边界；如果要改变上、下页边距，则可用鼠标指向垂直标尺上的页边距边界，待鼠标箭头变成双向箭头后拖动页边距边界。

② 用鼠标左键单击【文件】菜单中的【页面设置】命令，然后用鼠标左键单击【页边距】选项卡，在上、下、左、右的框中填入具体的数值，如图 5-24 所示。

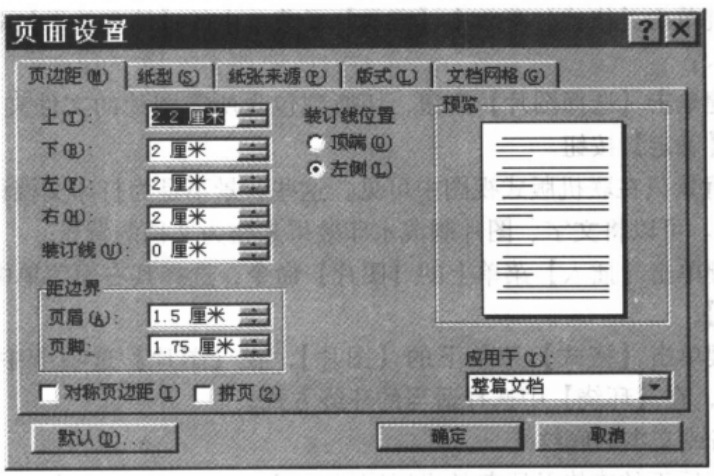


图 5-24　【页面设置】对话框

(3) 插入页码

在文档中插入页码时，可以控制页码的显示位置。其方法如下：

- 用鼠标左键单击【插入】菜单中的【页码】命令；

- 在【位置】框中选择【页面顶端】或【页面底端】等；
- 在【对齐方式】中选择【左侧】等。

(4) 创建页眉或页脚

页眉是指加入到页面顶端空白区域内的文本或图形。页脚是指加入到页面底端空白区域内的文本或图形。

- 用鼠标左键单击【视图】菜单中的【页眉和页脚】命令，弹出如图 5-25 所示工具栏；

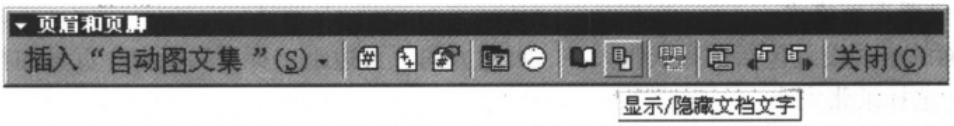


图 5-25 【页眉和页脚】工具栏

- 在要创建一个页眉区的位置输入文字或图形，也可用鼠标左键单击【页眉和页脚】工具栏上的第一个按钮；
- 要创建一个页脚，可用鼠标左键单击【在页眉和页脚切换】按钮以便移至页脚区，然后重复上一个步骤；
- 完成以上步骤后，用鼠标左键单击【关闭】按钮。

(5) 添加背景

可以在 Word 文档中添加各种各样的背景（如编织纹理），使其看起来更加生动活泼：

- 给 Word 文档添加背景色或纹理。用鼠标左键单击【格式】菜单中的【背景】命令，再单击所需颜色或单击【其他颜色】命令，查看其他可选择的颜色。用鼠标左键单击【填充效果】命令，可得到特殊效果（如纹理）。
- 在 Word 中用图片作为联机阅读的背景：

● 用鼠标左键单击【格式】菜单中【背景】子菜单中的【填充效果】命令，再单击【图片】选项卡；

● 用鼠标左键单击【选择图片】按钮，切换到包含所需图片的文件夹，选好一张图片，然后单击【确定】按钮。

以上选定的背景只在联机版式视图中可见，这些背景不是为打印文档设计的。

- 创建水印。可以将文字、图片制成水印效果来作为文档的背景。

● 用鼠标左键单击【插入】菜单上的【图片】命令，选择其右面菜单中的【剪贴画】或【来自文件】；

● 用鼠标左键单击【格式】菜单下的【图片】，在【图片】选项卡中选择颜色框中的【水印】项，在【环绕】中选择适当的环绕方式。

(6) 取消表格或文本的底纹

以下操作对于后面的表格及段落的设置同样适用。

- 要取消表格底纹，可用鼠标左键单击该表格中的任意位置；要取消指定单元格的底纹，可选定这些单元格；要取消段落的底纹，可用鼠标左键单击该段中的任意位置；要取消指定文字的底纹（如一个单词），则选定该文本。
- 用鼠标左键单击【格式】菜单中的【边框和底纹】命令，再单击【底纹】选项卡，

在【填充】下单击【无】。

3. 设置段落格式

Word 中的段落是指前一个“回车”之后到当前“回车”之间的所有字符。设置段落格式可选择【格式】菜单中的【段落】一项。其窗口如图 5-26 所示。

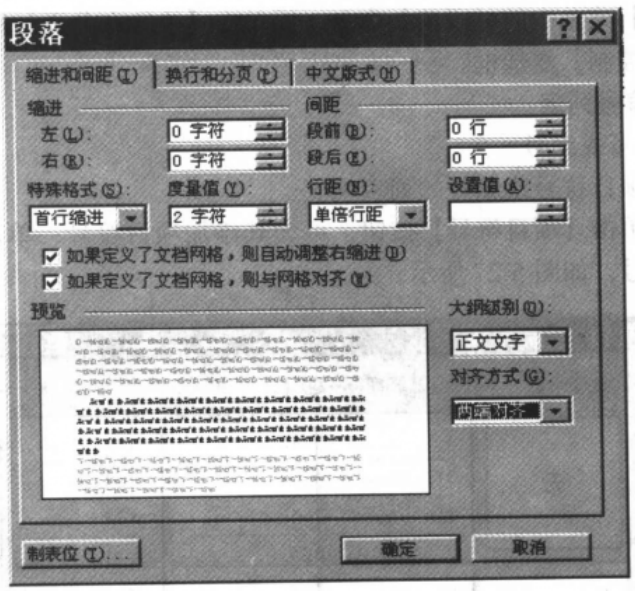


图 5-26　【段落】对话框

Word 显示的文本和打印出的文本是完全相同的。用鼠标左键单击【帮助】菜单中的【这是什么】命令可查看指定字符或段落的格式。在鼠标箭头变成问号后，用鼠标左键单击想查看的文本，即可显示其格式。查看完毕后，可按 Esc 键关闭。

(1) 设置行距

行距表示文本字符行与行之间的垂直间距。在默认情况下，Word 采用单倍行距。所选行距将影响所选段落中的所有文本行。如果某行包含大字符、图形或公式，则 Word 将自动增加行距。要使所有的行距相同，可用鼠标左键单击【行距】框中的【固定值】选项，然后在【设置值】框中键入能容纳最大字体或图形的行距。如果字符或图形只能显示一部分，则可在【设置值】框中选择更大的行距值。

(2) 段落缩进

增加或减少缩进量时，则需改变文本和页边距之间的距离；创建悬挂缩进时，则需定义后续行相对于文本第一行左侧的偏移量。

(3) 调整段落间距

- 选定需要改变段落间距的段落。
- 用鼠标左键单击【格式】菜单中的【段落】命令，再单击【缩进和间距】选项卡。
- 在【间距】选项的【段前】和【段后】框中键入所需间距值，可调节段前和段后的间距（“段前”指当前段第一行与前一段最后一行的距离；“段后”指当前段最后一行与下一段首行之间的距离）。

（4）孤行控制

孤行是指单独打印在一页顶部的某段落的最后一行，或者是单独打印在一页底部的某段落的第一行。Word 中的默认设置是防止出现孤行。

- 选定含有需要加以控制的孤行段落。
- 单击【格式】菜单中的【段落】命令，再单击【换行和分页】选项卡。
- 选中【孤行控制】复选框。

4. 项目符号和编号列表

（1）添加项目符号或编号

- 选定要添加项目符号或编号的列表项。
- 用鼠标左键单击【项目符号】按钮，可为其添加项目符号；单击【编号】按钮，可为其添加编号，如图 5-27 所示。

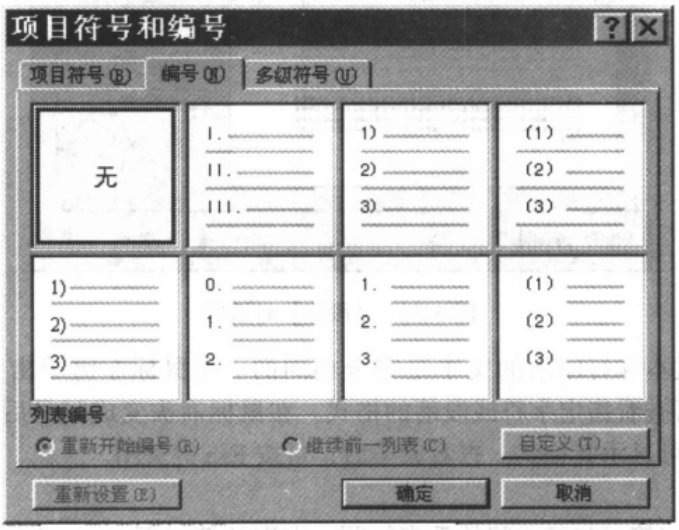


图 5-27 【项目符号和编号】对话框

（2）项目符号与编号的互相转换

- 选定要进行转换的项目或编号。
- 用鼠标左键单击【项目符号】按钮或【编号】按钮。

（3）更改项目符号或编号格式

- 选定含有要更改的项目符号或编号的段落。
- 用鼠标左键单击【格式】菜单中的【项目符号和编号】命令，再依据要更改的列表的类型单击相应的选项卡。
- 用鼠标左键单击【自定义】按钮。
- 选择所需格式选项。

（4）删除项目符号或编号

- 选定要删除的项目符号或编号项。
- 用鼠标左键单击【项目符号】按钮，可删除其项目符号；单击【编号】按钮，可删除其编号。

5.2.4　表格的基本操作

1. 创建表格

使用表格可组织信息，并可通过两栏并排放置文字和图形，建立引人入胜的版面。使用【插入表格】按钮可以快捷地创建一个简单的表格。

使用“绘制表格”功能可以如同用笔一样随心所欲地绘制复杂的表格。例如，表格各单元格的高度可以不同，各行可有不同的列数。

(1) 创建简单表格

- 用鼠标左键单击文本区要创建表格的位置。
- 用鼠标左键单击【插入表格】按钮或单击【表格】菜单下的【插入表格】项，选定所需的行数、列数，如图 5-28 所示。
- 用鼠标左键单击【确定】按钮完成。

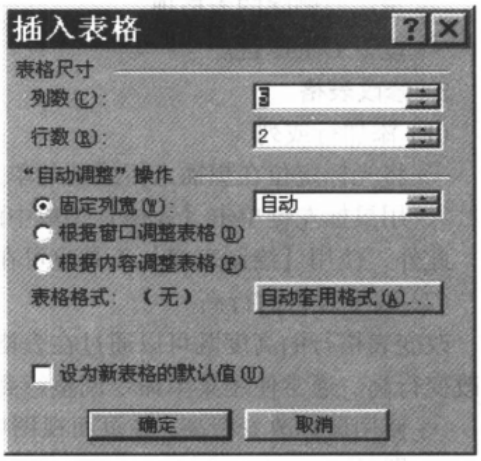


图 5-28　【插入表格】对话框

(2) 创建复杂表格

- 用鼠标左键单击文本区要创建表格的位置。
- 如果屏幕上没有【表格和边框】工具栏，则需用鼠标左键单击【表格和边框】按钮以显示该工具栏，再单击【绘制表格】按钮，指针可变为笔形（或单击【视图】菜单，再单击【工具栏】项中的【表格和边框】项）；如果屏幕上已显示【表格和边框】工具栏，则单击【绘制表格】按钮，指针可变为笔形，如图 5-29 所示。

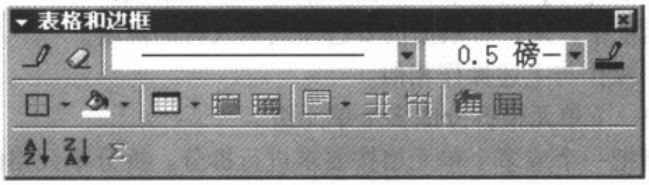


图 5-29　【表格】和【边框】工具栏

- 确定表格的外围边框，可以从表格的一角拖动到其对角，然后再绘制各行各列。如果要擦除边框线，则可用鼠标左键单击【擦除】按钮并在要擦除的框线上拖动。
- 创建表格后；用鼠标左键单击某个单元格，即可在该单元格中键入文字或插入图形。

(3) 移动或复制表格项

- 选定要移动或复制的单元格、行或列。
- 如果移动，则需拖动选定内容到新位置；如果复制，则在按住 Ctrl 键的同时拖动选定内容到新位置。

2. 取消表格或删除表格项

在 Word 文档中，用户可以删除单个或多个单元格、行或列，也可删除整张表格，还可以只删除单元格的内容而不删除单元格本身。

(1) 删除表格及其内容

- 用鼠标左键单击要删除的表格。
- 用鼠标左键单击【表格】菜单中的【删除表格】命令。

(2) 删除表格的单元格、行或列

- 选定要删除的单元格、行或列。
- 用鼠标左键单击【表格】菜单中的【删除单元格】、【删除行】或【删除列】命令。

(3) 删除表格内容

- 选定要删除的表格项。
- 按下 Delete 键。

3. 修改表格

(1) 添加行或列

- 将光标定位在想插入行或列的单元格上。
- 用鼠标左键单击【表格】菜单中的【行】或【列】命令。

此外，使用【绘制表格】工具也可在相应的位置绘制行或列。

(2) 改变表格的行高

改变表格行的高度既可以通过在表格中添加文本时改变行高，也可以使用鼠标或菜单命令改变行高。通常使用菜单命令调整行高更精确。

- 使用鼠标改变行高：在页面视图中使用鼠标拖动表格的行边框或垂直标尺上的标志即可。
- 使用菜单改变行高的具体操作如下：

- 选择要改变的一行或多行；
- 用鼠标左键单击【表格】菜单中【表格属性】命令，出现【表格属性】对话框；
- 用鼠标左键单击【行】选项卡，弹出一个【行】对话框；
- 在【行】对话框中，用户可根据需要选择或输入行的高度。

改变表格列宽的操作与上述操作相似。

(3) 将表格中的一个单元格拆分成多个单元格

拆分单元格是指把一个或多个单元格按要求进行拆分。拆分单元格既可以把一个单元格分成多个单元格，也可以对多个单元格同时进行拆分。其具体操作步骤如下：

- 选定要拆分的一个或多单元格；
- 用鼠标左键单击【表格】菜单中的【拆分单元格】命令，弹出对话框；
- 在【列数】文本框中输入要拆分的列数，在【行数】文本框中输入要拆分的行数；
- 用鼠标左键单击【确定】按钮即可完成拆分。

(4) 拆分表格

将一个表格拆分成两个表格的操作如下：

- 选定要拆分的表格；
- 用鼠标左键单击【表格】菜单中的【拆分表格】命令。

(5) 合并单元格

将同一行或同一列中的两个或多个单元格合并为一个单元格的具体操作如下：

- 选定要合并的多个单元格；

- 用鼠标左键单击【表格】菜单中的【合并单元格】命令，即可删除所选单元格之间的表格线，使其成为一个大的单元格。

4. 设置表格格式

(1) 自动设置表格格式

Word 系统提供了丰富的表格格式。用户可以直接套用这些格式：

- 选定要格式化的表格；
- 用鼠标左键单击【表格】菜单中【表格自动套用格式】命令；
- 在【格式】下用鼠标左键单击所需格式。

(2) 添加边框

在 Word 文档中，用户可以为表格、段落或选定文本的四周或任意一边添加边框，也可为文档页面四周或任意一边添加各种边框，包括图片边框。

- 把插入点移到要添加边框的表格。
- 用鼠标左键单击【格式】菜单中的【边框和底纹】命令，再单击【边框】选项卡。
- 根据对话框中的选项，选择所需的选项来完成边框设置。

(3) 单元格中文本的垂直对齐方式

- 用鼠标左键单击【表格和边框】按钮，显示【表格和边框】工具栏。
- 用鼠标左键单击要设置文本对齐方式的单元格。
- 在表格工具栏中对齐按钮中选择合适的对齐方式。

(4) 重复表格标题

- 选定要作为表格标题的一行或多行文字。
- 用鼠标左键单击【表格】菜单中的【标题行重复】命令。

注意： 重复的表格标题只能在【页面视图】中查看。

5. 表格的简单计算功能

在表格中，用户可以进行简单的运算。

(1) 引用表格中的单元格

表格中的单元格可用诸如 A1，A2，B1，B2 之类的形式进行引用。其中，字母代表列，数字代表行。在公式中引用不连续的单元格时，各单元格之间用逗号分隔，而引用连续或矩行区域的单元格时，首尾单元之间用冒号分隔。

(2) 计算行或列数值的总和

- 用鼠标左键单击要放置求和结果的单元格。
- 用鼠标左键单击【表格】菜单中的【公式】命令，显示公式对话框。
- 在该文本框中输入公式。例如，选定的单元格位于某列数值的底端，Word 将按公式 =SUM（ABOVE）进行计算。若计算正确，则用鼠标左键单击【确定】按钮。
- 用鼠标左键单击【确定】按钮即可完成计算。

注意：

① 如果 Word 正在显示域代码，则括号内看到的是域代码而不是求和结果。要显示域代码的计算结果，请用鼠标左键单击【工具】菜单中的【选项】命令，再单击【视图】选项卡，然后清除【域代码】复选框。

② 如果该行或该列中含有空单元格，则 Word 将不累加整行或整列。要对整行或整列求和，请在空单元格中键入零值。

③ 要快捷地对一行或一列数值求和，请先用鼠标左键单击要放置和的单元格，再单击【表格和边框】工具栏中的【自动求和】按钮。

5.2.5 简单的图文操作

1. 插入图片

在 Word 中，用户可从【剪辑库】中插入剪贴画或图片，也可以从其他程序和位置插入图片或扫描照片。

(1) 从【剪辑库】中插入剪贴画

- 将插入点定位于想插入剪贴画的位置。
- 在【插入】菜单中指向【图片】子菜单，用鼠标左键单击【剪贴画】命令，然后再单击【剪贴画】或【图片】选项卡。
- 用鼠标左键单击所需类型，然后用鼠标左键双击图像。

(2) 插入另一个文件中的图片

- 将插入点置于想插入图片的位置。
- 在【插入】菜单中指向【图片】子菜单，然后用鼠标左键单击【来自文件】命令。
- 找到包含所需图片的文件，用鼠标左键单击需要插入的图片。

2. 添加或编辑特殊文字效果

用【绘图】工具栏中的【艺术字】可以给文字增加特殊效果。例如，可以创建带阴影的、斜体的、旋转的和延伸的文字，还可创建符合预定形状的文字。因为特殊文字效果是图形对象，所以也可用【绘图】工具栏上的按钮来改变其效果（如可用图片填充文字效果）。需要特别强调的是，特殊文字效果是图形对象，不能作为文本对待。在大纲视图中无法查看其文字效果，也不能像普通文本一样检查其拼写。

(1) 增加特殊文字效果

- 用鼠标左键单击【绘图】工具栏上的【插入艺术字】按钮，弹出如图 5-30 所示【“艺术字”库】对话框。

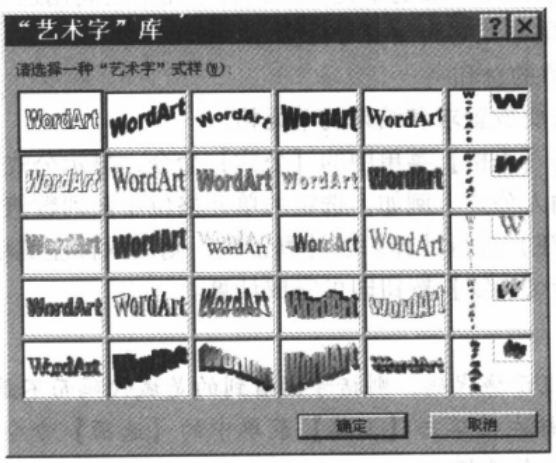


图 5-30 【“艺术字”库】对话框

- 选择所需的特殊效果，用鼠标左键单击【确定】按钮。
- 在【编辑‘艺术字’文字】对话框中，键入要设置格式的文字，选择所需的其他选项，用鼠标左键单击【确定】按钮。

(2) 旋转或翻转图形对象

用户可将图形对象（包括艺术字）右旋或左旋 90°或其他任何角度，也可水平或垂直翻转对象，还可翻转或旋转一个对象或一组对象。

- 选定要旋转的对象。
- 用鼠标左键单击【绘图】工具栏上的【自由旋转】按钮。
- 向要旋转的方向拖动对象的一角。

(3) 图形对象的三维效果

① 添加或改变图形对象三维效果，操作步骤如下：

- 选定需要修改的图形对象；
- 用鼠标左键单击【绘图】工具栏上的【三维效果】按钮，在弹出的菜单中，用鼠标左键单击所需选项后即可添加三维效果；如要改变三维效果，如颜色、旋转、深度、照明度或表面纹理，则用鼠标左键单击【三维效果】按钮，再单击【三维设置】命令，最后单击【三维设置】工具栏上的所需选项即可。

② 取消图形对象三维效果的操作步骤如下：

- 用鼠标左键单击需要修改的图形对象；
- 用鼠标左键单击【绘图】工具栏上的【三维效果】按钮，再单击【无三维效果】按钮。

③ 改变图形对象的三维效果颜色的操作步骤如下：

- 选定需要修改的图形对象；
- 用鼠标左键单击【绘图】工具栏上的【三维效果】按钮，再单击【三维设置】命令；
- 在【三维设置】工具栏上用鼠标左键单击【三维颜色】按钮旁边的箭头，再单击所需颜色。

3. 编辑图片

在对一张图片进行编辑之前，必须先选定想编辑的图片，然后选择【图片】工具栏上合适的选项进行编辑。

(1) 使文字环绕图形对象

- 在【页面视图】中，选定文本框或图形对象。
- 用鼠标左键单击【格式】菜单中的【图片】命令，再单击【版式】选项卡，如图 5-31 所示。
- 在【环绕方式】框中，选择文字与图形的环绕方式。
- 用鼠标左键单击【高级】按钮，弹出【高级版式】对话框，选择文字环绕选项卡，可以设置文字环绕方式及图片上边、下边、左边、右边与文字之间的距离。
- 连续两次用鼠标左键单击【确定】按钮，即可完成文字的环绕设置。

(2) 设置图片的颜色及亮度

- 选中要编辑的图片。
- 用鼠标左键单击【格式】菜单中的【图片】命令，再单击【图片】选项卡。

(3) 用鼠标左键单击【增加/降低对比度】按钮或【增加/降低亮度】按钮。

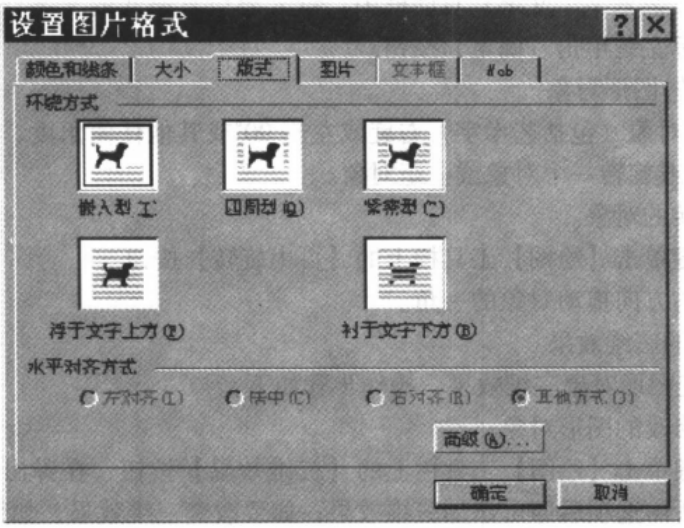


图 5-31 “设置图片格式”窗口

5.2.6 预览与打印

编辑文档完成后，用鼠标左键单击常用工具栏上的【打印】按钮即可将其打印出来。若希望在实际打印前预先看看将来的打印效果，则可用鼠标左键单击【打印预览】按钮即可显示打印后的各页样式。

1. 打印预览

如果需要查看对象（如页眉、页脚或图形对象）的打印效果，则可以切换至【页面视图】。用户可使用【打印预览】按钮查看文档打印后结果。在 100% 显示比例（可使用【视图】菜单中的【显示比例】对话框设置）下，屏幕显示最接近实际的打印结果。另外，如果在普通视图中编辑文档，则可以使 Word 精确地显示结果。在普通视图中，用鼠标左键单击【工具】菜单中的【选项】命令，再单击【视图】选项卡，清除【大纲视图和普通视图】选项下的【草稿字体】复选框。此外，在联机版式视图打印文档时，打印结果可能与显示结果不同。打印前切换至普通视图，可得到“所见即所得”的文档。

(1) 打印前预览文档

用鼠标左键单击【文件】菜单下的【打印预览】命令或单击常用工具栏上的【打印预览】按钮，可以显示打印后文档的外观。

(2) 在打印预览视图中编辑文本

- 在打印预览状态下显示需要编辑的页面，并用鼠标左键单击需要编辑范围中的文字，Word 会放大显示此区域。
- 用鼠标左键单击【放大镜】按钮，鼠标指针会由放大镜形状变成 I 形，这时即可开始修改文档。
- 用鼠标左键单击【放大镜】按钮，再单击文档，可以返回原显示比例。
- 用鼠标左键单击【关闭】按钮，可以退出打印预览视图并返回文档的原视图。

2. 打印文档

用鼠标左键单击常用工具栏的【打印】按钮即可将全部文档打印出来，也可以用鼠标左键单击【文件】菜单下的【打印】命令来对打印机及打印范围进行设置，然后再打印，如图 5-32 所示。

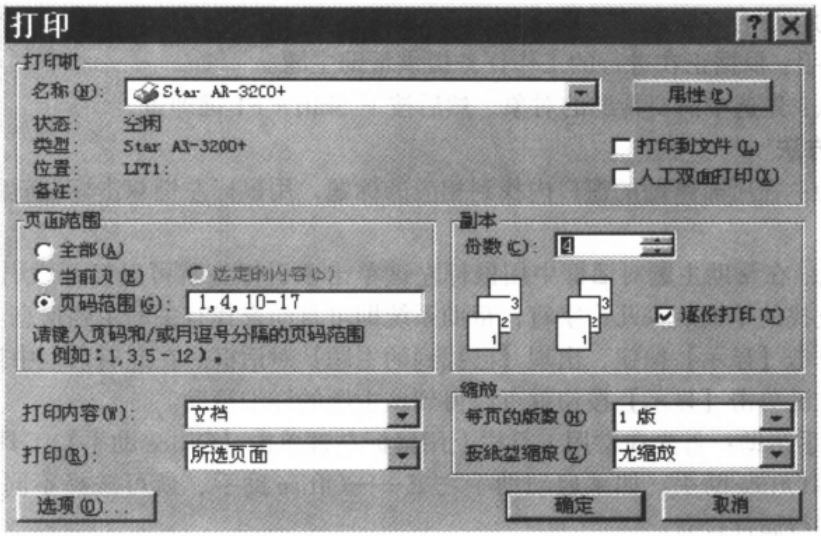


图 5-32　【打印】对话框

(1) 打印文档指定部分

- 在【页面范围】下指定文档需打印的部分。
- 如果用鼠标左键单击【页码范围】选项，则应该键入用逗号分隔开的页码或用减号分隔开的页码范围，或两者都键入。图 5-32 显示的是将要打印文档的第 1 页、第 4 页及第 10～17 页。

(2) 同时打印多份

- 在【份数】框中键入要打印的份数。
- 若选中【逐份打印】复选框，则在打印完整的一份副本后才开始打印下一份的第一页；消除此复选框，即可将所有副本的首页打印完毕再开始打印其他后继页。

(3) 只打印奇数页或者偶数页

用鼠标左键单击【打印】框中的【奇数页】选项或【偶数页】选项即可。

(4) 一次打印多篇文档

- 用鼠标左键单击【文件】菜单下的【打开】按钮。
- 在文件夹列表中找到并用鼠标左键双击包含所需打印文档的文件夹，选择所有要打印的文档。

- 选择【打开】，对话框中不相连的多个文件，先用鼠标左键单击一个文件名，按住 Ctrl 键后，再用鼠标左键单击其他文件名。
- 选择【打开】对话框中相连的文件，先用鼠标左键单击第一个文件名，按住 Shift 键后，再用鼠标左键单击最后一个文件名。

- 用鼠标左键单击【命令和设置】按钮，再单击【打印】按钮。

5.2.7 帮助系统的使用

1. 帮助的形式

索引帮助：是对想完成某项操作、但又不知需要如何操作的帮助。

目录帮助：适合初学者的帮助。

Office 助手：根据正在进行的工作任务提供帮助主题。

这是什么：先选中需要帮助的对象，然后按下 Shift + F1 键即可。

2. 使用方法

目录帮助：在所需帮助的窗口内找到相应的标题，用鼠标左键双击标题就能进入该标题的内容。

索引帮助：在帮助主题对话框中用鼠标左键单击索引标签就可以打开索引的选项清单，然后输入要查找的单词的头几个字符，帮助系统即可自动查找有关的主题信息。选中后，再用鼠标左键单击【显示】按钮，出现【已找到的主题】对话框。在此对话框中选择相应的主题，用鼠标左键单击【显示】按钮就可以得到该主题的详细帮助。

Office 助手帮助：首先在常用工具栏上用鼠标左键单击【Office 助手】工具栏或按下 F1 键，即可看到 Office 助手。如果启动助手之家——Office 助手，就可选择不同形式的助手，以及对助手进行属性设置。

5.3 Excel 操作

Excel 是微软公司推出的基于 Windows 98 环境的电子表格软件，能够准确、快捷地处理数据、制作表格和绘图。Excel 功能强大，所以应用非常广泛。

5.3.1 Excel 概述

1. Excel 的主要功能

(1) 制作与编辑表格

Excel 的主要功能是建立和编辑表格。这些表格被称为工作表。表中的每一个小格被称为单元格，用于存放数据。若干张相关的工作表可组成一个工作簿。

(2) 强大的计算功能

在 Excel 表格中允许使用公式和函数，可完成各种复杂的运算。

(3) 设置工作表的格式

Excel 能够对工作表中的行、列及单元格的字体、字型、边框进行修饰。

(4) 数据的管理与分析

Excel 提供了标准的数据库管理功能，如查询、排序、筛选、增加或删除记录及统计、汇总数据等，便于处理大量的数据。

2. Excel 的启动与退出

(1) 启动 Excel 的方法

Excel 的启动分为菜单启动、利用快捷图标启动及利用快捷工具栏启动等方法。

(2) 退出 Excel 方法

退出 Excel 之前应保存现有的工作文件，然后可以用以下三种方法退出：

- 菜单操作：打开【文件】菜单，用鼠标左键单击【退出】命令；
- 使用关闭按钮：用鼠标左键单击 Excel 窗口右上角的【关闭】按钮；
- 使用快捷键：按 Alt + F4 键也可以关闭当前窗口，退出 Excel。

3. Excel 的工作界面

Excel 的工作界面可以划分为功能区、工作区和状态栏三大部分，如图 5-33 所示。

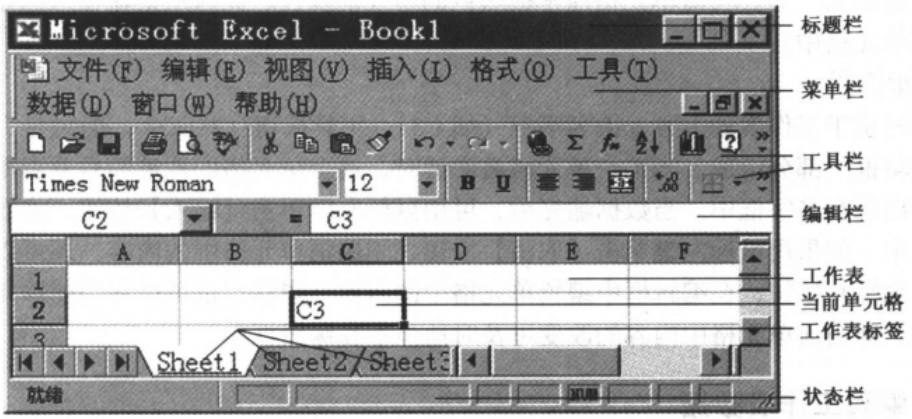


图 5-33　Excel 工作界面

(1) 功能区

功能区在屏幕的上部，包括标题栏、菜单栏、工具栏及编辑栏。前三者的功能及使用方法与 Word 窗口中的相应部分类似。

(2) 工作区

工作区是一个很大的表，但屏幕中只能显示出其中的一部分表格。工作区包括行号、列号、滚动条、工作表标签及工作表标签滚动按钮。

(3) 状态栏

状态栏在屏幕的下部。其功能是显示当前工作状态或提示用户进行适当的操作。

4. Excel 的基本概念

(1) 工作簿与工作表

工作簿就是在 Excel 环境中用来存储并处理工作数据的文件。一个工作簿文件就是一个 Excel 文档。其默认名称为 Book n，默认扩展名为 .XLS。当启动 Excel 时，就自动打开了一个工作簿。在默认状态下，每个工作簿会打开 3 个工作表。根据用户的需要，在一个工作簿中最多可以建立 225 个工作表，其中只有一个是当前工作表。工作表的名称显示在工作簿窗口下部的工作表标签队列里，用鼠标左键单击标签队列中工作表的名称可以切换当前工作表。工作表的默认名称为 Sheet n。若要更改工作表名称，则用鼠标右键单击工作表标签，再用鼠标左键单击【重命名】命令即可。

(2) 单元格与单元格区域

工作表可分为若干行和列，行号用数字表示，从 1～65 536，共 65 536 行；列号用字母表示，从 A～IV 列，共 256 列（排列方式为 A～Z，AA～ZZ，BA～BZ，…，IA～IV）。每一对行、列坐标所指向的位置上的方格被称为单元格。所以，一张工作表中最多能有

65536×256个单元格。每个单元格占据工作表中惟一的位置。表示单元格位置的列编号、行编号就是单元格的地址（称为列地址、行地址，书写时先列后行）。为了区别不同工作表中同一位置上的单元格，必要时可以在单元格地址前加工作表名称，如 Sheet1！A1。

实际上，一个单元格就是一个存储数据的单元，数据可以是一个字符、一组数字、一个公式或图形、声音等。当用户对工作表进行操作（如输入数据、修改或复制数据）时，需要逐个处理单元格，正在被操作的单元格被称为“当前单元格”或者“活动单元格”、“当前单元”、“活动单元”等。当前单元格通常被黑色的粗线框围住，此时从键盘输入的数据可以显示在当前单元格中。

(3) 编辑栏

编辑栏位于工作表的上方，由名字框（如 C3）、工具按钮（共三个，即对号、错号、等号）和编辑框三部分组成。当某个单元格被激活时，这个单元格的地址编号或者单元格区域的名称将出现在名字框中。当数据输完后，可用鼠标左键单击【输入】按钮，将数据存入当前单元格中。如果用鼠标左键单击【取消】按钮，则取消单元格中的内容。还可以用鼠标左键单击【编辑】按钮，在编辑框中编辑单元格中的内容。当然，也可以在当前单元格中进行编辑修改，即当前单元格中内容的改变可及时地显示在编辑框中。

5.3.2 编辑工作表数据

1. 新建工作簿与打开工作簿

(1) 新建工作簿

启动 Excel 时，系统将自动打开一个工作簿。建立新的工作簿有三种方法：

- 用鼠标左键单击【文件】菜单中的【新建】命令；
- 用鼠标左键单击常用工具栏上的【新建】按钮；
- 在 Windows 98 桌面上单击鼠标右键，并在出现的快捷菜单中选择【新建】→【Microsoft Excel 工作簿文件】命令。

(2) 打开工作簿

① 在 Microsoft Excel 中，可以打开保存在本机硬盘或有可连接的网络驱动器上的工作簿。其具体操作步骤如下：

- 用鼠标左键单击【打开】按钮；
- 在【查找范围】框中，用鼠标左键单击包含所需工作簿的驱动器、文件夹；
- 在文件夹清单中，用鼠标左键双击各层文件夹，直到打开包含所需工作簿的文件夹；
- 用鼠标左键双击需要打开的工作簿。

如果需要打开一个最近使用过的文件，请在【文件】菜单的底部用鼠标左键单击此文件名。如果近期使用过的文件没有被列出，请用鼠标左键单击【工具】菜单中的【选项】命令，再单击【常规】选项卡，然后选中【最近使用的文件清单】复选框。

② 在 Microsoft Excel 中，还可以一次打开多个工作簿。其操作步骤如下：

- 用鼠标左键单击【打开】按钮，在【查找范围】框中，用鼠标左键单击包含所需工作簿的驱动器、文件夹，在文件夹清单中，用鼠标左键双击各层文件夹，直到打开包含所需工作簿的文件夹；
- 如果需要选中几个不相邻的文件，请先用鼠标左键单击其中一个文件名，然后按住

Ctrl 键，再用鼠标左键单击其余的文件名；如果需要选中几个相邻的文件，则先用鼠标左键单击序列中的第一个文件名，然后按住 Shift 键，再用鼠标左键单击最后的文件名；

- 用鼠标左键单击【打开】按钮。

如果选中了一个不需要的文件，则按住 Ctrl 键，然后再次用鼠标左键单击该文件名。

同时打开多个文件时，系统会打开多个窗口，每个窗口中显示一个工作簿文件，但是只有一个窗口是当前工作簿窗口。可以用鼠标左键单击选择或利用窗口菜单项选择当前工作簿文件。

(3) 在工作簿中选定工作表

在打开的工作簿中必须选定某个工作表为当前工作表才能向其中输入数据。其方法是：把鼠标指针移到窗口下方的工作表标签队列上，用鼠标左键单击指定的工作表标签。当前工作表的标签用白底表示，未选取的工作表标签为灰色底色。

(4) 选定当前单元格

向工作表中输入数据应总是在当前单元格中进行。每一个单元格可以保存一个数据。在使用工作表时，需要在这些单元格中选择当前单元格，这一操作被称为移动单元格指针。Excel 中单元格指针的移动有以下多种方式。

① 在显示范围内移动。如果目的单元格在当前的显示区域中，则将鼠标指向目的单元格，然后在其上用鼠标左键单击或双击即可。如果要指定的单元格不在当前显示区域中，则把鼠标箭头移到屏幕滚动条的箭头上，把所需单元格显露出来，然后再用鼠标左键单击此单元格，使其成为围着粗黑框的当前单元格。

② 利用【名字框】。可以在编辑栏的【名字框】中输入目的单元格地址（如 B2），然后按 Enter 键即可。

③ 使用定位命令。在【编辑】菜单中选择【定位】命令，在弹出的对话框中的【引用位置】框中输入目的单元格的地址，然后用鼠标左键单击【确定】按钮，则目的单元格即成为当前单元格，并显示在屏幕中。

(5) 选择单元格区域

有时，操作不是对一个单元格进行，而是对由若干个单元格组成的一个区域进行的，如删除、插入、移动、复制若干列、若干行等，这就必须首先选定要操作的单元格区域。

选择单元格区域有如下几种方法。

① 选取整个工作表。有时需要选取整个工作表，以便对工作表进行全面的修改，如清除全部内容、改变所有字符格式或改变字体大小等。在这种情况下，只需用鼠标左键单击工作表左上角的小格——【选取整个工作表】按钮，则可选择整个工作表。

② 选定整行或整列（以选定整列为例）。选定整列的操作比较简单，只需在工作表上用鼠标左键单击该列的列号即可。如果要选取连续的多列，则可以按住 Shift 键用鼠标左键单击逐个列号即可选定。

③ 选取一个矩形区域。选取一个矩形区域的方法有如下三种。

- 鼠标操作。先将鼠标指向该区域的第一个单元格，按住鼠标左键，沿着对角线拖动鼠标到该区域右下角位置上的最后一个单元格，松开鼠标左键即可完成一个矩形区域的选定，所选取的范围将反白显示，如图 5-34 所示。

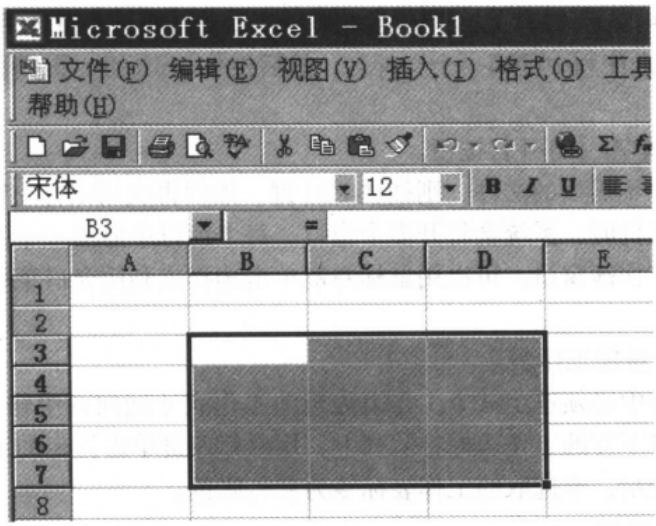


图 5-34 选择区域

- 键盘操作。选取一个矩形区域，可先将单元格指针移到选取范围的左上角，然后按住 Shift 键，同时按方向键进行选取。

有时，需要几个不连续的单元格或区域进行操作，即需要选定不相邻的区域。其方法是按住 Ctrl 键，然后用鼠标左键单击所需的单元格、行、列，或用上述鼠标拖动的方法选定若干不连续的区域。

- 使用【编辑】菜单中的【定位】命令，在引用位置中分别写上左上角及右下角的单元格地址，中间用冒号分开，如 B3:D6。

(6) 保存工作簿文件

用鼠标左键单击常用工具栏中的【保存】按钮，或者选择【文件】菜单下的【保存】命令项，都会出现要求输入文件名以保存文件的对话框，这时可以命名工作簿文件并把工作簿文件存盘。应注意，Excel 是以工作簿文件（*.XLS）为存储单位的，不能把其中的一张工作表作为文件保存。

2. 输入数据建立工作表

(1) 输入数据

选定单元格或区域后，工作表下面的状态栏中即会显示“就绪”，等待用户输入数据。在向单元格输入数据时，单元格内会出现闪动的编辑光标，状态栏中显示“输入”，同时编辑栏中出现上述的三个工具按钮。

如果在输入状态下发现输入错误，则可按 Backspace 键或 Delete 键修改。未输完数据之前，不要按方向键或 Enter 键；否则，系统会认为已输完，单元格指针后移到下一单元格，无法继续输入所余数据。输完数据后，用鼠标左键单击【输入】按钮或其他单元格，或者按 Enter 键，则数据存入单元格，下一个单元格成为带黑粗框的活动单元格，Excel 回到就绪状态。

① 在工作表单元格中输入数据的方法：首先用鼠标左键单击需要输入数据的单元格，然后键入数据并按 Enter 或 Tab 键。

② 同时在多个单元格中输入相同数据的方法如下：

- 请选定需要输入数据的单元格，选定的单元格可以是相邻的，也可以是不相邻的；
- 键入相应的数据，然后按 Ctrl + Enter 键。

② 同时在多张工作表中输入相同数据的方法如下：

- 在选定了一组工作表后，在活动表选定单元格区域中进行的任何改动都将反映到所有其他工作表中对应的单元格，这些工作表中的相应数据将被替换。
- 选定需要输入数据的工作表，再选定需要输入数据的单元格或单元格区域。
- 在第一个选定单元格中键入或编辑相应的数据，然后按 Enter 键或 Tab 键，Excel 将自动在所有选定工作表的相应单元格中输入相同的数据。

(2) 修改单元格数据

几乎所有的操作在未完成时都可以按 Esc 键取消。对于已经存入单元格的数据可用以下方法修改：

① 用鼠标左键单击单元格，状态栏显示“输入”，可以直接输入新的内容，同时原内容被删除。如果按 Delete 键，则删除其中的全部内容。

② 用鼠标左键双击单元格，状态栏显示“编辑”，当前单元格中又出现闪动的编辑光标(竖线形)，而且编辑栏中的三个工具按钮也再度出现。此时可以按方向键、Backspace 键或 Delete 键修改单元格中原有的数据。

(3) 为单元格加批注

在表格中能够显示出来的数据是有限的，有时候需要更详细地说明单元格中的数据，即给单元格加批注。其方法是用鼠标右键单击欲加批注标记的单元格，在弹出的菜单中选择【插入批注】，或在选定单元格后，用鼠标左键单击【插入】菜单项下的【批注】，在弹出的方框内输入要附加说明的内容，然后用鼠标左键单击【确定】按钮，如图 5-35 所示。

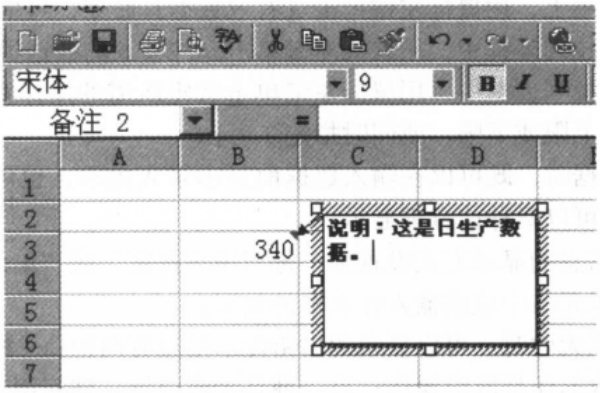


图 5-35　插入【批注】窗口

取消批注的方法是用鼠标右键单击带批注标记的单元格，在弹出的菜单中选择【删除批注】，当然也可以通过【编辑】菜单中的【清除】选项进行删除。

(4) 填充柄的使用

选定单元格右下角的小黑块叫填充柄。将鼠标指向填充柄时，鼠标的形状变为黑“十”字形。通过拖动单元格填充柄，可以将选定单元格中的内容复制到同行或同列中的其他单元格。如果该单元格中包含 Microsoft Excel 可扩展序列中的数字、日期或时间段，则在这一操

作过程中，这些数值将按某种规律填充到这些单元格中去。

① 在同一行或列中复制数据，操作方法如下：

- 选定包含需要复制数据的单元格；
- 用鼠标左键拖动选定单元格右下角的填充柄经过数据的单元格，然后释放鼠标左键，即可完成复制数据。

复制的单元格数据将替换被填充单元格中原有的数据或公式。如果在填充过程中，类似于数字或日期等数值发生增长而不是照原样复制，则需重新选定原始数据，然后按住 Ctrl 键，再拖动填充柄。

② 填充数字、日期或其他序列，操作方法如下：

- 选定待填充数据区的起始单元格，然后输入序列的初始值。如果要让序列按给定的步长增长，则需再选定下一单元格，在其中输入序列的第二个数值。头两个单元格中数值的差额将决定该序列的增长步长。
- 选定包含初始值的单元格。
- 用鼠标拖动填充柄经过待填充区域。如果要按升序排列，请从上向下或从左到右填充；如果要按降序排列，请从下向上或从右到左填充。

(5) 为单元格指定有效数据的范围

如果希望确认工作表上输入数据的正确性，则可以为单元格或单元格区域指定输入数据的有效范围。将数据限制为特定的类型，如整数、小数或文本，并且限制其取值范围。此外，还可以通过工作表中的数据序列来指定有效数据的范围或限定允许输入的字符数。其具体方法如下：

- 选定希望限制其有效数据范围的单元格；
- 在【数据】菜单上，用鼠标左键单击【有效数据】命令，再单击【设置】选项卡；
- 在【许可】下拉列表框中，用鼠标左键单击允许输入的数据类型；
- 在【数据】下拉列表框中，用鼠标左键单击选定所需的操作符，再根据选定的操作符指定数据的上限或下限，或同时指定二者；
- 在检查有效数据时，既可以在输入数据前显示录入提示，也可以在数据输入后显示出错提示，还可以两种信息都显示。

可以将录入提示信息的显示方式设置为“每当用户选定”或“单击单元格时”即进行显示。该信息用来解释单元格中应该输入什么样的数据；

当用户输入的数据无效时，可以显示的出错提示信息有两种：一种是提示信息，让用户选择是保留已输入的数据还是取消操作；另一种是警告信息，询问用户是确认输入有效并继续其他操作，是取消操作或是返回并更正数据。

如果希望阻止无效的数据输入到单元格中，则可以选择最严格的一种方式，显示出错提示信息，并在错误被更正或取消前禁止用户继续工作。其具体操作是：在【样式】下拉列表框中，用鼠标左键单击【中止】按钮，如图 5-36 所示。

(6) 建立公式

作为一个电子表格系统，Excel 除了能进行一般的表格处理外，更主要的是对数据的计算功能。在表格处理中，经常需要对表格中的数据进行计算和统计，对此，用户只要给出相应的计算公式或指明统计要求，剩余的工作就可以全部由计算机自动完成。

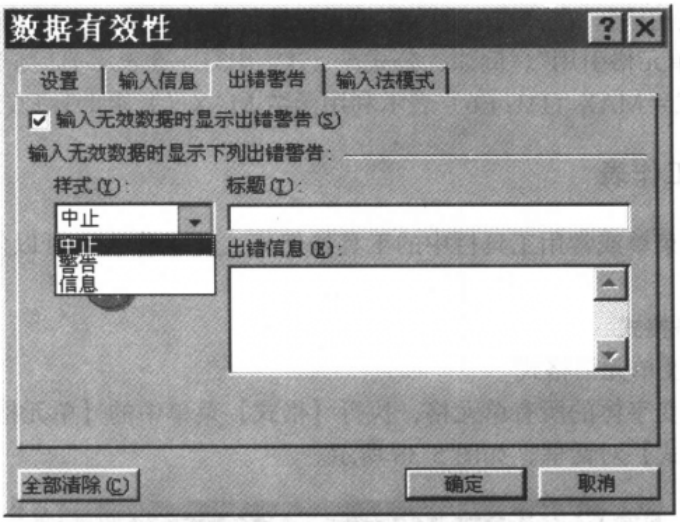


图 5-36　选择【中止】项

Excel 中的公式都必须用符号“=”或者“+”来引导。这是公式单元格与一般只存放数据的单元格最大的区别。在如图 5-37 所示中，在 A2 单元格中输入 30，在 B2 单元格中输入公式“= A2 × 10”。输入公式后，用鼠标左键单击编辑栏中的【输入】按钮，或者按 Enter 键确认。输入了公式的单元格被称为“公式单元格”。若公式正确，则单元格将显示计算结果，如公式单元格 B2 中将出现计算结果 300。以后，只要用鼠标左键双击公式单元格，则公式将同时出现在公式单元格及编辑框中供用户修改。

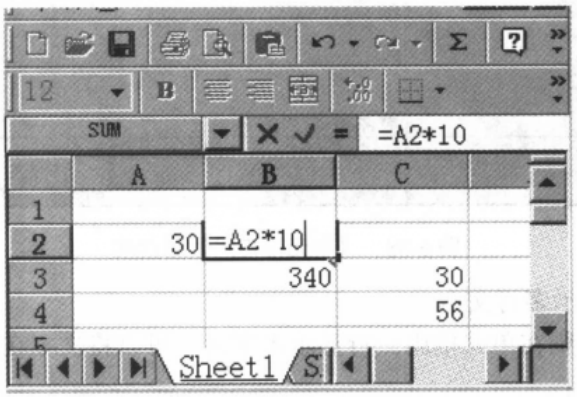


图 5-37　公式单元格

Excel 公式中的运算符。使用运算符可以对公式中所包含的数据进行特定类型的运算。Excel 包含四种类型的运算符，即算术运算符、比较运算符、文本运算符（&）及引用运算符。

Excel 公式中除了运算符外，还可以使用以下成分：

- 允许使用多层圆括号，计算顺序为先算内层再算外层；
- 允许使用 Excel 提供的内部函数；
- 允许使用常数，即固定不变的数据，如 −64，280，−8，−90，“hello”等；

- 允许单元格引用。在公式中允许将单元格地址作为变量，引用其中的数据参加运算。这称为“单元格引用”。例如，公式＝（B2－B3）＊5 表示 B2 与 B3 单元格值之差的 5 倍；公式＝MAX（D3：E6）表示利用函数 MAX（）求 D3：E6 区域中的最大值。

5.3.3 格式化工作表

使用【格式】菜单或常用工具栏中的工具按钮格式化工作表，可以使工作表重点突出、清楚，容易理解。

1. 设置单元格格式

(1) 设置文本及单元格格式

首先选定要改变字体的所有单元格，执行【格式】菜单中的【单元格】命令，这时屏幕上出现【单元格格式】对话框，如图 5-38 所示。

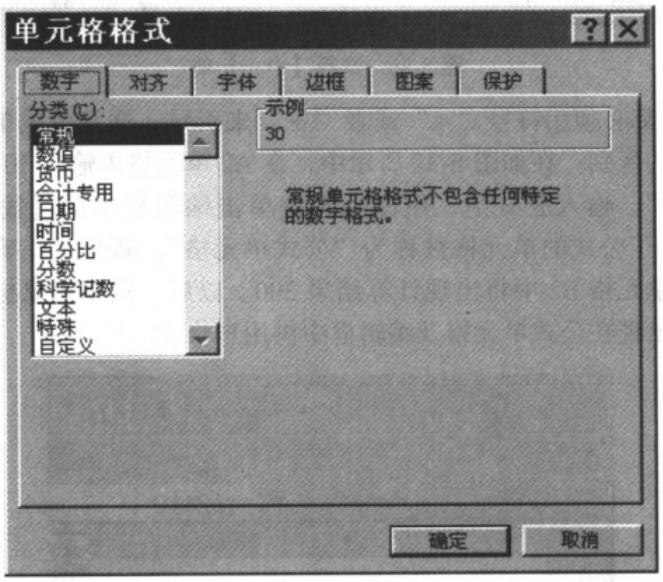


图 5-38 【单元格格式】对话框

(2) 在单元格内显示多行文本

若一行文本太长，则在单元格内一行显示不下时，可以将文本在一个单元格内拆成多行显示。其操作步骤如下：

- 选择要设置格式的单元格，用鼠标左键单击【格式】菜单中的【单元格】命令，然后单击【对齐】选项卡；
- 在文本控制的下面，选中【自动换行】复选框。

此外，要在指定位置开始新的一行，可在编辑栏（或正编辑的单元格）中要换行的地方用鼠标左键单击一下，然后按下 Alt＋Enter 键。

(3) 复制一个单元格或单元区域的格式

若某单元格已设定好格式，则要将该格式应用到其他单元格中的操作步骤如下。

- 选择含有要复制格式的单元格或单元区域，用鼠标左键单击常用工具栏上的【格式刷】按钮；

- 选择要设置新格式的单元格或单元格区域。

此外，若要将选定单元格或区域的格式复制到多个位置，则需用鼠标左键双击【格式刷】按钮。当完成复制格式时，再用鼠标左键单击该按钮。

2. 应用边框、底纹和背景

(1) 设置单元格的边框

在初始状态下，Excel 建立工作表时表格无边框线（通过打印或打印预览可看出）。在工作薄窗口中，单元格边框显示为淡灰色。不过，用户可以自行设置边框线，特别是必须将工作表以表格的形式打印出来时尤为必要。

- 选择要设置格式的单元格。用鼠标左键单击【格式】菜单中的【单元格】命令，选择【边框】选项卡，显示如图 5-39 所示。

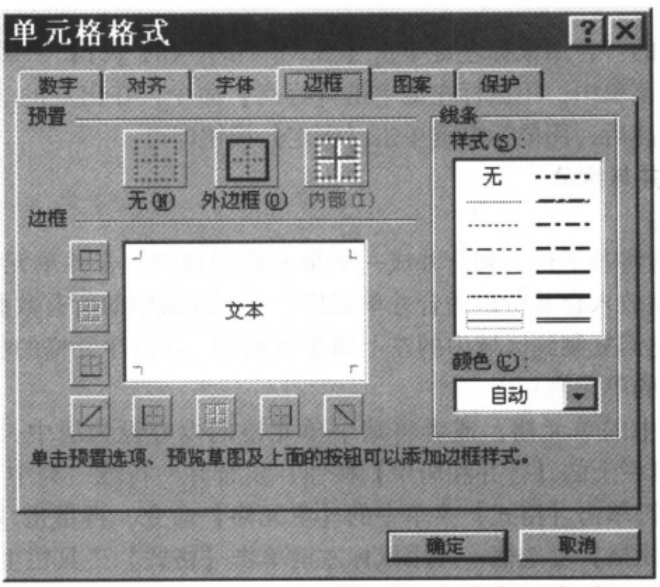


图 5-39　单元格边框

- 用鼠标左键单击所需的线型样式，再单击对话框中间文本区的相应位置。
- 用鼠标左键单击【确定】按钮完成。

(2) 设置单元格背景色

- 选择要设置背景色的单元格。
- 用鼠标左键单击如图 5-38 所示中的【图案】选项卡。要设置带有图案的背景色，可用鼠标左键单击【单元格底纹】框。
- 用鼠标左键单击【图案】框，再单击所需的图案样式和颜色。
- 若要删除单元格背景色，则只需选择要删除背景色的单元格，并用鼠标左键单击【无颜色】按钮即可。

3. 更改列宽与行高

用户输入的数据有时很长且单元格放不下时，则太长的文本可能显示不完全，其中的数字则可能显示为“＃＃＃＃＃＃＃＃＃”；有时数据又很短，浪费空间。为此需要改变行高和列宽，以完整显示单元格的数据，并能使工作表清楚明了。

(1) 用鼠标改变行高和列宽

用鼠标改变行高和列宽的方法有以下两种方法（以改变列宽为例）。

- 将鼠标移动到要改变的列的列头右边界上，鼠标的指针变为一个用竖线连起的左右箭头的形状。拖动该边界向右，则增大列宽；拖动该边界向左，则减小列宽。
- 用上面介绍的方法使鼠标变成左右箭头的形状后，用鼠标左键双击右边界，Excel 将自动调整列宽到能放下该列的最长的数据为止。

当要同时改变多列时，则可先选定那些列，然后拖动其中的任意一列的边界，即可同时调整这些列，且它们的宽度相同。

(2) 使用菜单改变行高和列宽

使用【格式】菜单项或快捷菜单也可以调整行高和列宽。以改变行宽为例的菜单方式的操作方法如下：

- 选定要调整的行，用鼠标左键单击【格式】菜单项的【行】子菜单；
- 在出现的子菜单中，用鼠标左键单击【行高】命令，在出现的【行高】对话框中填入需要的行高后，用鼠标左键单击【确定】按钮即可。

4. 单元格的合并与拆分

(1) 合并单元格

合并单元格是指将若干行、列合并成一个单元格的过程。合并单元格后，Excel 只把选定区域左上角的数据放入合并所得的合并单元格。要把区域中的所有数据都包括到合并后的单元格中，必须将它们复制到区域内的左上角单元格中。合并单元格的步骤如下：

- 选择要合并的单元格；
- 要合并一行中的单元格，或者将某个单元格的内容设为居中，则用鼠标左键单击【格式】工具栏上的【合并和居中】按钮；要合并一行或一列中选定的几个单元格，则用鼠标左键单击【格式】菜单中的【单元格】命令，再单击【对齐】选项卡，选中【合并单元格】复选框，可用鼠标左键单击【格式】工具栏上的【左对齐】、【居中】或【右对齐】来更改文本的对齐方式。

(2) 拆分单元格

- 用鼠标左键单击已合并的单元格，选择【格式】菜单中的【单元格】命令，然后再单击【对齐】选项卡。
- 清除【合并单元格】复选框。

5. 打印预览与打印

中文 Excel 由于采用了“所见即所得”的技术，因此一旦准备好要打印的数据时，就可以预览打印结果，并且可以调整页面的设置来得到所要的输出结果。

使用打印预览能同时看到全部页面，并可调整像分页符和页边距之类的内容。在通常情况下，Excel 对工作表的显示同打印后所看到的工作表在形式上是一致的。在打印之前，最好先保存工作薄。这样，即使打印机产生错误或发生其他问题，也不会丢失最后一次保存工作薄之前所完成的工作。

5.3.4　数据管理和分析

1. 数据排序

为了方便查看数据，往往要对数据进行排序。排序是根据某一（几）列的数据按一定的规律排列记录。

(1) 使用排序工具按钮对数据进行排序

在常用工具栏中有两个排序工具按钮：一个是【升序】按钮，选择它将按从小到大的顺序排序；另一个是【降序】按钮，选择它将按从大到小的顺序排序。

当仅仅需要对工作表中的某一列数据进行排序时，则只需要用鼠标左键单击此列中的任一单元格，再单击常用工具栏中的【升序】按钮或【降序】按钮即可。

(2) 使用菜单命令对数据进行排序

使用【升序】或【降序】按钮排序只能对一列的数据进行排序，并且操作不太方便。有时需要对多个列进行排序，此时需使用【数据】菜单项的【排序】命令。其方法：先选定需要排序的单元格，再选择【数据】菜单项的【排序】命令，将出现如图 5-40 所示的【排序】对话框。在此对话框中最多可以设定三个层次的排序标准，即主要关键字、次要关键字及第三关键字。其中，【当前数据清单】选项在这里是指选定的区域是否包括标题行。最后用鼠标左键单击【确定】按钮即可完成排序。



图 5-40　“排序”对话框

2. 数据筛选

用户可能经常希望按照某些条件从数据库中选取一部分数据。这就需要使用数据筛选命令。Excel 中提供了“自动筛选”和“高级筛选”两种筛选方式。在此只介绍“自动筛选”。

使用自动筛选功能可以将工作表中符合条件的记录显示出来，并隐藏不符合条件的记录。下面以学生成绩单为例介绍自动筛选功能。其具体过程如下。

↘ 选定字段名区域，如图 5-41 所示。



图 5-41　自动筛选

- 用鼠标左键单击【数据】菜单项的【筛选】子菜单中的【自动筛选】命令，Excel 将在工作表中的每个字段名的旁边加入一个下拉箭头按钮，并将出现一列表框，在列表框中列出了可以筛选的项目。其各项目的含义如下。

- 全部：取消该字段的筛选，显示全部记录；
- 前 10 个…：主要用于数值字段，通过对话框可以设置筛选出最大（或最小）的若干条记录；
- 自定义：如果选择此项，则将出现“自定义自动筛选方式”对话框。在此对话框中可以自定义筛选条件，为一个字段指定两个筛选条件，然后按照这两个筛选条件的组合进行数据筛选。

如果选择该字段的所有出现过的值中的某一个值，那么将筛选出等于该值的所有记录。

3. 制作图表

将数据以图表形式显示，可以令数据更清楚、更易于理解。图表有助于分析数据、比较不同数据之间的差异。用户可以建立一个独立图表以展示数据，也可以将图嵌入到工作表中。

(1) 建立图表

Excel 可以建立两种图表：一种是嵌入图表，即在工作表中嵌入图，需同时显示数据和图表时，可以建立嵌入图作为工作表中的对象；另一种是在工作簿中现有工作表之外再建立一张独立的图表作为一张特殊的工作表，这样，图表与数据是分开的，可以分别打印。但无论是哪种图表，它们的依据都是工作表中的数据，当工作表中的数据改变时，图表会自动更新。图表的制作步骤如下。

- 选定用于制作图表的数据。制作图表之前，须先在工作表中选择一区域。该区域中的数据将是制作图表的依据。选择区域可以是连续的，也可以是不连续的。如果区域中的第一行为文本，并且最左列也为文本，则若想将它们标注在图上，则选定数据区域时应将它们选择在内。
- 启动【图表向导】建立图表。选定工作表区域后，选择【插入】菜单中的【图表】命令。启动【图表向导】后，将出现如图 5-42 所示的【图表向导】对话框。在【图表类型】列表框中列出了多种图表类型。在对话框的右部列出了当前图表类型的几个子图表类型。应先选择图表类型，再选择子图表类型。

图表类型的选择是很重要的，选择一个能最佳表现数据的图表类型，可有助于更清楚地反映数据的差异和变化。以后的步骤可按照【图表向导】对话框中的提示即可完成。

(2) 打印图表

嵌入的图表属于这个工作表。其打印与打印工作表的方法一致。打印独立图表，则需经过如下步骤。

- 选定要打印的图表，调整图表的大小比例。
- 从【文件】菜单中选择【页面设置】命令，并在【页面设置】对话框中选择【图表】选项卡。在【打印图表大小】框中选择【调整】选项；在【打印质量】框中选择【按黑白方式打印】复选框（如果是彩色打印机，则不必设置此项）。
- 用鼠标左键单击【打印】按钮，出现【打印】对话框，再单击【确定】按钮即开始打印。

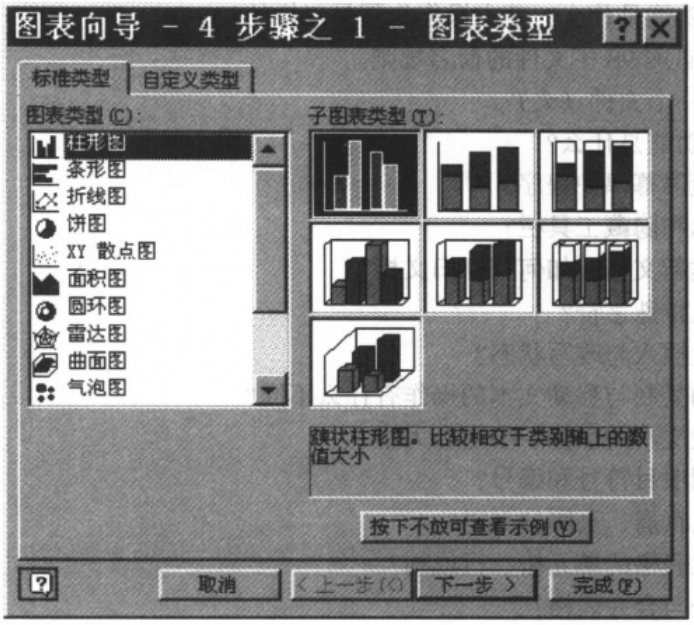


图 5-42　【图表向导】对话框

小结

Windows 98 操作系统、文字处理软件 Word 和电子表格软件 Excel 是微软公司推出的视窗操作系统和办公系列软件。由于具有优秀的图形界面、简单方便的操作方式，故使其成为个人计算机中应用最广泛的系列软件。

本章主要讲述了 Windows 98 操作系统、文字编辑软件 Word 和电子表格软件 Excel 等软件的使用和操作方法。

在 Windows 98 操作系统部分，主要讲述的是 Windows 98 操作系统的基本特点和桌面的基本构成，需要重点掌握的是在 Windows 98 操作系统下如何管理和操作文件及各种资源。

在文字处理软件 Word 部分，主要讲述的是文字处理软件 Word 的基本特点和功能，并重点介绍了文字处理软件 Word 的启动与退出、Word 窗口的基本构成、文档的基本编辑、文档的格式化、表格的基本操作、简单的图文操作及文档的打印与预览等功能，很多功能的实现讲解了几种操作方法，应熟练掌握其中的一种方法。

在电子表格处理软件 Excel 中，主要介绍 Excel 的特点、窗口的构成、编辑工作表中的数据、格式化工作表、数据管理及分析等内容。

习题

1. 简述 Windows 98 的主要功能特点。
2. 简述 Windows 98 桌面的基本组成元素及功能。
3. 鼠标的基本操作有哪几种?
4. 试述 Windows 98 的窗口和对话框的组成元素。
5. 在 Windows 98 中，运行应用程序有哪几种方式? 最常用的是什么方式?

6．剪贴板的功能是什么？基本操作步骤是怎样的？

7．试述 Windows 98 中文件的命名规则。

8．如何复制一个文件（夹）？

9．回收站的功能是什么？

10．如何设置屏幕保护程序？

11．怎样显示和隐藏工具栏？

12．如何创建新文档？如何打开旧文档？

13．如何插入特殊字符？

14．怎样切换插入与改写状态？

15．使用拖动复制与移动文本的操作有什么不同？

16．如何设置文本的字体、字形和大小？

17．如何使用项目符号和编号？

18．如何插入页眉、页脚和页码？

19．如何插入表格的单元格、行或列？

20．如何拆分与合并表格？

21．如何插入剪贴画、图形文件？

22．如何设置图文环绕？

23．打印预览的功能是什么？

24．打印时需进行哪些参数的设置？

25．如何启动和关闭 Excel？

26．在 Excel 中，“清除”单元格和“删除”单元格的意义和操作有什么不同？

27．如何对工作表进行改名？如何在工作簿中插入新工作表？

28．如何在一个工作簿中复制或移动一张或几张工作表？如何在不同工作簿中复制或移动一张或几张工作表？

29．若已经在 Excel 中输入了一个等差数列，则现在要将这个数列改为等比数列，应该如何操作？

30．连续数据的自动求和及非连续数据的自动求和，在操作上有什么不同？

31．在 Excel 中，通过工作表可以生成的图表主要有什么类型？

32．如何在数据清单中进行自动筛选？

33．什么是分类汇总的分级显示？如何改变这种分级显示的显示级别？

第 2 篇

C 语言程序设计

第 6 章　C 语言概述

本章要点

- ☑ C 语言的发展历程
- ☑ C 语言的特点
- ☑ C 源程序的基本结构

6.1　C 语言简介

6.1.1　C 语言的发展历程

C 语言是在 B 语言的基础上发展起来的。它的发展与操作系统 UNIX 密切相关。在 C 语言诞生以前，UNIX 的早期版本是用汇编语言编写的。由于汇编语言程序依赖于计算机硬件，其可读性、可移植性都很差，编程困难。因此，1970 年，美国贝尔实验室的 Ken Thompson 设计出了简单且又接近硬件的 B 语言，并用 B 语言编写了 UNIX 操作系统。由于 B 语言依赖于计算机，过于简单，功能有限，又无数据类型，所以没有流行起来。1972 年至 1973 年间，贝尔实验室的 D.M.Ritche 在 B 语言的基础上设计出了 C 语言。C 语言既保持了 B 语言的特点（精练、接近硬件），又克服了它的缺点。1973 年，Ken Thompson 和 D.M.Ritche 合作，把 UNIX 中 90%以上的代码用 C 语言改写。后来，对 C 语言进行了多次改进，1977 年出现了不依赖具体计算机的 C 语言版本。随后，C 语言迅速普及，成为当今广泛使用的计算机语言之一。

20 世纪 80 年代初，美国国家标准化协会（ANSI）根据 C 语言问世以来各种版本对 C 语言的发展和扩充，制定了 ANSI C 标准（1989 年再次做了修订）。目前，流行的 C 语言版本都是以 ANSI C 标准为基础的版本。各种版本的基本部分相同，只在个别地方稍有差异。在 PC 机上经常使用的有 DOS 下的 Microsoft C、Turbo C 等。

在 C 语言的基础上，1983 年，贝尔实验室的 Bjarne Strou-strup 又推出了 C++。C++ 进一步扩充和完善了 C 语言，成为一种面向对象的程序设计语言。

6.1.2　C 语言的特点

C 语言同时具有汇编语言和高级语言的优势。因此，C 语言又被认为是中级语言，广泛用于编写应用程序和系统程序。C 语言的主要特点如下。

- 语言简洁、紧凑，使用方便、灵活。C 语言有丰富的数据类型和运算符，数据结构描述能力及表达式能力强，且使用关键字少（32 个），语言组成精炼、简洁，语法

限制不太严格，而且使用方便灵活。程序书写形式自由，压缩了一切不必要的成分，几乎允许任何类型之间的转换。

- 运算符极其丰富。C语言提供了40种运算符，包括位运算符，可以完成其他高级语言难以完成的运算。
- 生成的目标代码质量高，程序执行效率高。一般只比汇编程序生成的目标代码效率低10%～20%，所以执行效率高。
- 可移植性好（与汇编语言相比），即容易从一种类型的计算机系统移植到另一种类型的计算机系统。
- 可以直接操纵计算机硬件。C语言可直接访问物理地址，能进行位（bit）操作，能实现汇编语言的大部分功能，可以直接对硬件进行操作。

6.2 C源程序的基本结构

下面先看几个简单的C语言程序，再认识和体会C语言源程序的基本结构和特点。

[例6-1] 编写C语言程序，打印一行字符。

其程序如下：

```
main()
{
  printf("I WISH YOU SUCCESS");
}
```

运行这个程序后，计算机将输出两个双引号中间的一行字符串“I WISH YOU SUCCESS”。printf（）是函数库中的一个函数。此处是用于调用库函数来完成输出功能。

[例6-2] 求两个数之和。

其程序如下：

```
main()                    /* 求两个数之和 */
{
   int a,b,sum;           /* 定义两个变量 */
   a=123;b=234;
   sum=a+b;
   printf("sum is %d\n",sum);
}
```

运行这个程序后，计算机将计算两个数之和，然后输出结果。

[例6-3] 输入a、b、c三个整型数，输出其中的最大者。编C语言程序实现。

```
#include "stdio.h"                          /* 编译预处理:标准输入输出头文件 */
main()
{
  int smax();
  int a,b,c,max ;
  scanf("%d,%d,%d",&a,%b,&c);               /* 标准输入函数调用 */
```

```
    max=smax(a,b,c);
    printf("a,b,c= %d,%d,%d,max= %d",a,b,c,max); /* 标准输出函数调用 */
}
int smax(x,y,z)                                  /* 用户自定义函数 */
  int x,y,z;                                     /* 形式参数说明 */
  {
    int k;
    if(x>y) k=x;
    else  k=y;
    if(z>k) k=z;
    return(k);
}
```

本程序包括两个函数 main() 和 smax()。在函数 main() 中，前两行是说明部分，后面的多行是可执行的语句部分。在函数 smax() 中，在花括号前是形式参数 x、y 和 z 的类型说明，花括号中第二行是变量的类型说明，后面四行是可执行的语句部分。

通过以上几个例子，可以看到：

- 程序的总体结构。一个 C 源程序是由一个主函数或者再加上若干个函数段组成的。程序的开始部分可以根据程序的需要写出以“#”为首字符的编译预处理行和全程序使用的变量说明，其后是若干函数段。每个段的最前面是函数名。主函数名是 main，其他函数段由程序员命名。函数名后是由圆括号“()”括起来的形式参数表。圆括号的后面是每一个形式参数的类型说明。再后就是由花括号“{}”括起来的函数主体了。
- 函数段内的内容。最简单的函数段是空函数段，执行它什么也不做。一般来说，根据程序功能的需要，函数段内可以有两部分内容，即说明部分和可执行部分（语句部分）。说明部分中可以包括多个说明项，用以说明函数中使用的变量和需调用的其他函数等。语句部分可以包括多个可执行语句。
- 各语句之间用分号“;”分隔开，并列的标识符或项之间用逗号“,”分隔；两个关键词相邻时，中间用空格（至少一个）分隔。
- C 语言在书写时是比较自由的，几个说明项或几个语句可以写在一行，一个语句可以分为几行写，但是一个词或一个数不能分两行写。写什么格式主要考虑程序易读和便于程序的维护。
- 为了使程序易读，在程序中设置注释部分，注释的内容写在“/*”和“*/”之间，程序运行时系统忽略其间的内容。

小结

C 语言既是国际上广泛流行的计算机高级语言，又是一种通用的结构化的程序设计语言。无论是系统软件还是应用软件及数据处理、数值计算等都可以很方便地使用 C 语言，如 Oracle、UNIX、Windows 等许多著名的软件都是用 C 语言开发的。

本章包括 C 语言的发展历程和 C 源程序的基本结构两部分内容。

在C语言发展历程部分，主要介绍的是C语言的发展历史和C语言的特点。

在C源程序部分，通过对几个简单C源程序的分析，介绍了C语言程序的基本组成，为后续章节打下良好的基础。

习题

1. 请根据自己的认识，写出C语言的主要特点。
2. C语言的主要用途是什么？它和其他高级语言有什么异同？
3. 写出一个C程序的构成。
4. 请参照本章例题，编写一个C程序，输出如下信息：

```
==========================================================
                        Good Morning!
==========================================================
```

第 7 章　算法及其描述方法

本章要点

- ☑ 算法的基本概念
- ☑ 常用算法举例
- ☑ 算法描述方法：自然语言、传统流程图、N-S 流程图、伪代码、计算机语言

7.1　算法概述

设计程序归根到底就是要让计算机帮助人们完成一项工作。为了达到这一目的，作为人指挥计算机的语言序列——程序，就必须包括两个核心问题，即“做什么”与“怎么做”。程序设计的任务就是要解决这两个问题。

对于“做什么”的问题，要在程序设计的前半阶段完成，就是要确定程序的功能，这对于大型程序是至关重要的。而对于“怎么做”的问题，那就要针对具体问题设计算法了。

计算机可以解决多类信息处理的问题，但人们必须先用计算机能够理解的语言（即程序设计语言）来详细描述解决问题的步骤，即首先要进行程序设计。对稍复杂一些的问题，要直接写出能解决该问题的计算机程序是困难的。为了克服这一困难，人们把程序设计的任务分成两步来完成：第一步，不使用程序设计语言而使用一种简单明了的表达方式设计出解决给定问题的算法；第二步，根据设计好的算法使用某种程序设计语言编写出对应于该算法的程序。第二步的工作要比第一步的工作容易得多。因此，算法设计是使用计算机解决问题过程中一个极为重要的环节。

粗略地说，解决问题的步骤序列就是算法。利用计算机解决问题需要有算法，在日常生活中做任何事情也都有它的算法。例如，烹调书中告诉人们烘烤面包的原料与操作步骤，这就是烤面包的算法；从北京开汽车去天津的路线走向就是从北京开车去天津的算法。如编写“从宿舍起床到教室上课”的“程序”，则程序功能如下：8:00 以前进入教室坐好，为了做到这一点，就要对 8:00 以前的“活动（算法）”进行设计，因此必须做到：7:20 前离开宿舍；7:50 前离开食堂；8:00 前进入教室。

这就是一个简单的算法。当然，我们关心的是用于计算机的算法，它们一般要复杂得多。

任何一个能够解决问题的算法都必须具备以下五个特性。

- 可执行性。算法中的每一个步骤都是可执行的。显然，这是一个正确算法必须具备的性质。例如，若在解决某问题的步骤序列中，有一个步骤“到商店去替我买支圆珠笔”是计算机无法执行的，则这一解决问题的步骤序列就不是算法。

- 确定性。算法中的每一个步骤均必须是明确定义的，不得有任何歧义性（非确定性）。例如，步骤“用 10 与 2 进行算术运算”就是一个有歧义的步骤，而步骤“计算 10 与 2 的和”则是具有确定性的步骤。如果一个解题的步骤序列中有的步骤是不明确的，那么使用这样的解题步骤序列就不能保证获得问题的准确答案。自然，这样的解决步骤序列不能称为算法。
- 有穷性。一个算法必须在执行有穷步之后结束。如果一个解题步骤序列永远不能结束，则就永远得不到问题的解答。因此，有始无终的解题步骤序列决不是算法。
- 有输入信息的说明。有的算法可以没有输入信息，然而大多数算法都具有输入信息。一个正确的算法要输出成品，而不加工对象提出的要求，则自然难以得到合格的成品。
- 有输出信息的步骤。既然算法是用来解决给定的问题的，那么一个正确的算法就必须应将人们所关心问题的答案输出来。因此，一个算法应当至少有一个输出问题答案的步骤。

为了获得易读、易懂、易修改、易扩充的算法，人们经过长期的实践，认为必须采取以下三项措施：

- 利用自顶向下的方法设计算法；
- 可利用顺序、分支、循环 3 种基本结构来构造算法；
- 要有优美的算法表达风格。

7.2 常用算法的举例

本节用 3 个典型的例子来说明在程序设计中如何设计具体算法，希望读者能够举一反三，灵活地对要解决的问题设计出最佳算法。

[例 7-1] 求 $1\times2\times\cdots\times9\times10$ 的积。

第一种算法：

可以用最原始的方法进行。其具体步骤如下。

(1) 先求 1×2，得到结果 2；

(2) 将 (1) 得到的乘积 2 再乘以 3，得到结果 6；

(3) 将 6 再乘以 4，得到结果 24；

(4) 将 24 再乘以 5，得到结果 120。

……

(5) 将 362880 再乘以 10，得到结果 3628800，即是最后结果。

虽然这样的算法是正确的，但过于繁锁。如果求 $1\times2\times\cdots\times10000$ 的积，则要写出 9999 个步骤，显然是不可取的，而且每次都直接使用上一步骤的数值结果（如 2，6，24，120 等），也不方便，故应当找到一种通用的表示方法。通过分析可以发现规律，即乘数是前一个数加 1。利用这一规律，可得到第二种算法。

第二种算法：

可以设两个变量，一个变量代表被乘数。另一个变量存放乘积结果，而直接将每一步骤的乘积放在被乘数变量中。现在，设 p 为被乘数，i 为乘数，用循环算法来求结果，则可以将算法改写如下：

(1) 使 $p=1$;

(2) 使 $i=2$;

(3) 使 $p\times i$，乘积仍放在变量 p 中，可表示为 $p\times i\rightarrow p$;

(4) 使 i 的值加 1，即 $i+1\rightarrow i$;

(5) 如果 i 不大于 10，则返回重新执行 (3)、(4) 及 (5)；否则，算法结束。

最后得到的 p 值就是 10! 的值。

仔细分析这一算法能否得到预期的结果。显然，第二种算法比第一种算法简练。

如果将题目改为求 $1\times3\times5\times7\times9\times11$ 的积，则算法只需做很少的改动即可：

(1) $1\rightarrow p$;

(2) $3\rightarrow i$;

(3) $p\times i\rightarrow p$;

(4) $i+2\rightarrow i$;

(5) 若 $i\leqslant11$，则返回 (3) 继续执行；否则，算法结束。

可以看出，用这种方法描述的算法具有通用性、灵活性。(3) ～ (5) 可组成一个循环，在实现算法时，要反复多次执行 (3)、(4)、(5)，直到某一时刻，执行 (5) 时经判断，乘数 i 已超过规定的数值而不返回 (3) 时为止。此时算法结束，变量 p 的值就是所求的结果。

由于计算机是具有高速运算能力的自动机器，实现循环是轻而易举的，所有计算机高级语言中都有实现循环的语句。因此，上述算法不仅是正确的，而且也是计算机能够实现的较好的算法。

[例 7-2]　120 个学生，要求将他们之中成绩在 60 分以上者打印出来。

用 n 表示学生学号，n_1 代表第一个学生学号，n_i 代表第 i 个学生学号。用 g 表示学生成绩，g_i 代表第 i 个学生的成绩。其算法如下：

(1) $1\rightarrow i$;

(2) 如果 $g_i\geqslant60$，则打印 n_i 和 g_i；否则，不打印；

(3) $i+1\rightarrow i$;

(4) 如果 $i\leqslant120$，则返回 (2) 继续执行；否则，算法结束。

本例中，变量 i 作为下标。用它来控制序号（第几个学生，第几个成绩）。当 i 超过 120 时，表示已对 120 个学生的成绩处理完毕，算法结束。

[例 7-3]　一个大于或等于 3 的正整数，判断它是否为一个质数。

所谓质数，是指除了 1 和该数本身之外，不能被其他任何整数整除的数。例如，13 是质数，因为它不能被 2，3，4，…，12 整除。判断一个数 n ($n\geqslant3$) 是否为质数的方法很简单，即将 n 作为被除数，将 $2\sim n-1$ 各个整数依次作为除数，如果都不能被整除，则 n 为质数。其算法如下：

(1) 输入 n 的值;

(2) $i=2$;

(3) n 被 i 除，得余数 r;

(4) 如果 $r=0$，表示 n 能被 i 整除，则打印 n “不是质数”，算法结束；否则，执行 (5);

(5) $i+1\rightarrow i$;

(6) 如果 $i\leqslant n-1$，则返回 (3)；否则，打印 n “是质数”，然后结束。

实际上，n 不必被 $2\sim n-1$ 之间的所有整数除，需被 $2\sim\sqrt{n}$ 之间的整数除即可。例如，判断 13 是否为质数，只需将 13 被 2、3 除即可。如果都除不尽，则 n 必为质数，(6) 可改为：如果 $i\leqslant\sqrt{n}$，则返回 (2)；否则，算法结束。

在考虑算法时，应当仔细分析所需判断的条件，如何一步一步地缩小被判断的范围。有些问题，判断的先后次序是无所谓的，而有些问题，判断条件的先后次序是不能任意颠倒的，应根据具体问题做具体分析。

通过以上几个例子，可以初步了解如何设计一个算法。

7.3 算法的描述方法

描述算法可以使用多种方法，常用的有自然语言、传统流程图、N-S 流程图、伪代码及计算机语言等。

7.3.1 用自然语言描述算法

前面介绍的算法是用自然语言描述的。自然语言就是人们日常使用的语言，可以是汉语、英语或其他语言。用自然语言描述算法通俗易懂，但文字冗长，容易出现“歧义”。自然语言描述算法表示的含义往往不太严格，要根据上下文才能判断其正确的含义。假如有这样一句话“王先生对李先生说他的孩子考上了大学。”，则请问是王先生的孩子考上了大学，还是李先生的孩子考上了大学呢？仅仅从这句话本身难以判断。此外，用自然语言描述算法包含分支和循环结构的算法，很不方便（如 7.2 节例 7-3 的算法）。因此，除了很简单的问题以外，一般不用自然语言描述算法。

7.3.2 用传统流程图描述算法

传统流程图可用一些图形表示各种操作。用图形描述算法，直观形象，易于理解。美国国家标准协会（American National Standard Institute，ANSI）规定了一些常用的流程图符号，如图 7-1 所示。这些符号已被世界各国程序员普遍采用。

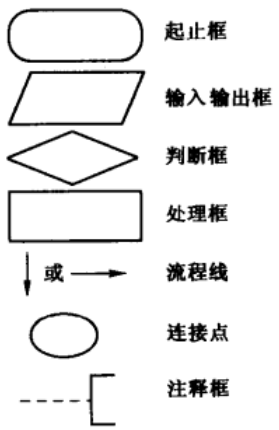


图 7-1 流程图常用符号

图 7-1 中菱形框的作用是对一个给定的条件进行判断，根据给定的条件是否成立来决定如何执行其后的操作。它有一个入口，两个出口，如图 7-2 所示。

连接点（小圆圈）用于将画在不同地方的流程线连接起来。图 7-3 中有两个以上以○为标志的连接点（在连接点圈内写上数字）。它表示这两个点是互相连接在一起的。实际上，它们是同一个点，只是画不下才分开来画的。连接点可以避免流程线的交叉或过长，可使流程图更加清晰。

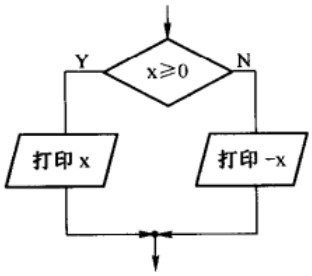


图 7-2　分支结构流程图

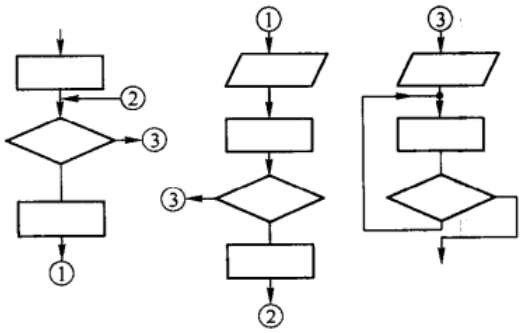


图 7-3　连接点应用示例

注释框是传统流程图中必要的部分。它不反映流程和操作，只是为了对流程图中某些框的操作做必要的补充说明，以帮助阅读流程图的人更好地理解流程图的作用。

[例 7-4]　将例 7-1 求 10! 的算法用传统流程图表示。其流程图如图 7-4 所示。

菱形框两侧的“Y”和“N”代表的是“是（Yes）”和“否（No）”。

通过这个例子可以看出，传统流程图是描述算法的较好工具。一个传统流程图包括几个部分：表示相应操作的框；带箭头的流程线；框内外必要的文字说明。

需要提醒大家的是，流程线不要忘记画箭头，因为它是反映流程执行的先后顺序的，如不画出箭头，就难以判断各框的执行次序了。

用传统流程图表示算法直观形象，可以比较清楚地显示出各个框之间的逻辑关系。前一时期，国内、外计算机书刊都广泛使用这种流程图表示算法。但是，这种流程图占用篇幅较多，尤其当算法比较复杂时，画流程图既费时又不方便。虽然在结构化程序设计方法推广之后，许多书刊已用 N-S 结构化流程图代替这种传统的流程图，但是每一个程序编制人员都应当熟练掌握传统流程图，做到会看会画。

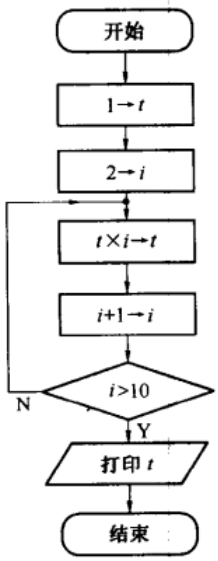


图 7-4　例 7-1 的传统流程图

7.3.3　用 N-S 流程图描述算法

1. 传统流程图的局限性

传统流程图用流程线指出各框的执行顺序，对流程线的使用没有严格的限制。因此使用者可以不受限制地使流程随意地转来转去，使流程图变得毫无规律。阅读者要花大量的精力去追踪流程，使人难以理解算法的逻辑。

为了提高算法的质量，使算法的设计和阅读更加方便，必须限制箭头的滥用，即不允许无规律地使流程随意转向，只能顺序地进行下去。但是，算法中难免会包含一些分支和循环，而不可能全部由一个一个框顺序组成。为了解决这一问题，人们设想，规定出几种基本结构，然后由这些基本结构按一定规律组成一个算法结构（如同用一些基本预制构件来搭成房屋一样），整个算法的结构由上至下地将各个基本结构顺序排列起来。如果能做到这一点，

则算法的质量就能得到保证和提高。

2. 三种基本结构

1966 年，Bohra 和 Jacopoini 提出了以下三种基本结构，并用这三种基本结构作为表示一个良好算法的基本单元。

(1) 顺序结构

如图 7-5 所示，虚线框内是一个顺序结构。其中，A 和 B 两个框是顺序执行的，即在执行完 A 框所指定的操作后，必然接着执行 B 框所指定的操作。顺序结构是最简单的一种基本结构。

(2) 选择结构

选择结构又称选择或分支结构，如图 7-6 所示。虚线框内是一个选择结构。此结构中必然包含一个判断框。根据给定的条件 p 是否成立而选择执行 A 框或 B 框。例如，p 条件可以是“$x \geqslant 0$”或“$x > y$”、“$a + b < c + d$”等。注意，无论 p 条件是否成立，均只能执行 A 框或 B 框之一，不可能既执行 A 框又执行 B 框。无论走哪一条路径，均在执行完 A 框或 B 框之后经过 b 点，然后离开本选择结构。A 框或 B 框中可以有一个是空的，即不执行任何操作，如图 7-7 所示。

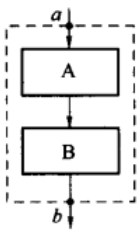


图 7-5 顺序结构

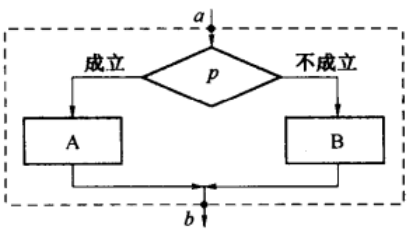


图 7-6 选择结构

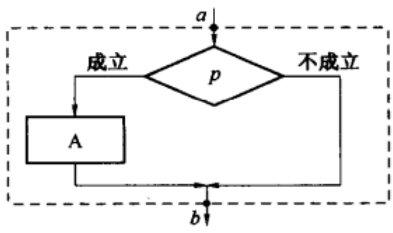
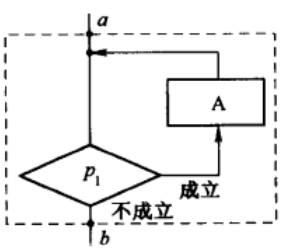


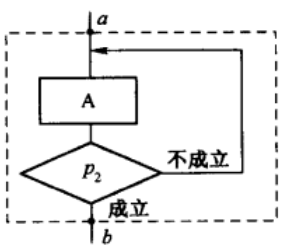
图 7-7 一分支为空的选择结构

(3) 循环结构

循环结构又称重复结构，即反复执行某一部分操作，可细分为两类循环结构：一类是当型（while 型）循环结构，如图 7-8（a）所示。它的功能是当给定的条件 p_1 成立时，执行 A 框操作，A 框执行完毕后，再判断条件 p_1 是否成立，如果仍然成立，则再执行 A 框，如此反复执行 A 框，直到某一次 p_1 条件不成立为止。此时不执行 A 框，而从 b 点离开循环结构。另一类是直到型（untill）循环结构，如图 7-8（b）所示。它的功能是先执行 A 框，然后判断给定的 p_2 条件不成立，再执行 A 框，然后再对 p_2 条件做判断，如果 p_2 条件仍然不成立，则继续执行 A 框，如此反复执行 A 框，直到给定的 p_2 条件成立为止。此时不再执行 A 框，从 b 点离开本循环结构。



(a) 当型循环结构



(b) 直到型循环结构

图 7-8 循环结构

以上三种基本结构具有的共同特点如下

- 只有一个入口，如图 7-5～图 7-8 所示中的 a 点为入口点。
- 只有一个出口，如图 7-5～图 7-8 所示中的 b 点为出口点。

注意：一个菱形判断框有两个出口，而一个选择结构只有一个出口。不要将菱形框的出口和选择结构的出口混淆。

- 结构内的每一个部分都有机会被执行到，即对每一个框来说，都应当有一条从入口到出口的路径通过它。
- 结构内不存在“死循环”（无终止的循环）结构。

已经证明，由以上三种基本结构顺序组成的算法结构，可以解决任何复杂的问题。由基本结构所构成的算法属于“结构化”的算法。它不存在无规律的转向，只在基本结构内才允许存在分支和向前或向后的跳转。

其实，基本结构不一定只限于上面三种，凡是具有上述 4 个特点的都可以作为基本结构。由它们构成的算法结构也是结构化的算法，可以认为这些结构都是由上述三种基本结构派生出来的。因此，人们普遍认为最基本的是本节介绍的三种基本结构。

3. N-S 流程图描述算法

既然基本结构的顺序组合可以表示任何复杂的算法结构，那么基本结构之间的流程线就属于多余了。

1973 年，美国学者 I.Nassi 和 B.Shneiderman 提出了一种新的流程图形式。在这种流程图中，完全去掉了带箭头的流程线，全部算法写在一个矩形框内。在该框内还可以包含其他的从属于它的框，或者说，有一些基本的框。这种流程图又被称为 N-S 结构化流程图（N 和 S 是两位学者的英文姓名的第一个字母）。因其适合于结构化程序设计，故很受欢迎。

N-S 流程图使用以下的流程图符号。

(1) 顺序结构：如图 7-9 所示。A 和 B 两个框组成一个顺序结构。

(2) 选择结构：如图 7-10 所示。它与如图 7-6 所示相对应。当 p 条件成立时，执行 A 操作；p 不成立时，则执行 B 操作。注意，图 7-10 是一个整体，代表一个基本结构。

(3) 循环结构：当型循环如图 7-11 所示，当 p_1 条件成立时，反复执行 A 操作，直到 p_1 条件不成立为止；直到型循环结构如图 7-12 所示。

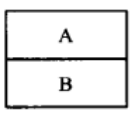


图 7-9　顺序结构

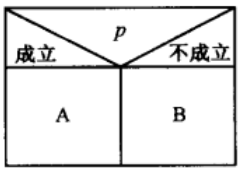


图 7-10　分支结构

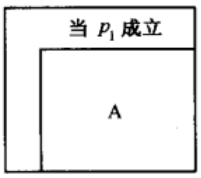


图 7-11　当型循环

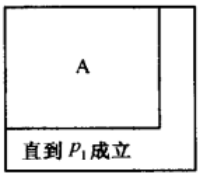


图 7-12　直到型循环

使用以上三种 N-S 流程图中的基本框，可以组成复杂的 N-S 流程图以表示算法。

应当说明，如图 7-9、图 7-10、图 7-11、图 7-12 所示中的 A 框或 B 框，可以是一个简单的操作（如读入数据或打印输出等），也可以是 3 个基本结构之一。例如，图 7-9 所示的顺序结构，其中的 A 框可以是一个选择结构，B 框又可以是一个循环结构，由 A 框和 B 框这两个基本结构组成一个顺序结构。

通过下面的例子，读者可以了解如何用 N-S 流程图表示算法。

[例 7-5] 将例 7-3 判别质数的算法用 N-S 流程图表示。

根据例 7-3 的描述，这个算法不能分解为三种基本结构，就无法直接用 N-S 流程图的三种基本结构的符号来表示。为了解决这个问题，设置一个标志变量 w，它的初值为 0。当 $r=0$ 时，不直接输出 n “不是质数”（表示 n 为非质数）使 w 变为 1；如果 $r\neq0$，则保持 $w=0$。w 的作用如同一个开关，有两个工作状况，即 $w=0$ 和 $w=1$。$w=1$ 表示 n 为非质数。如果最终 $w=0$，则表示 n 为质数，再使用一个分支结构输出 n 是否是质数，如图 7-13 所示。

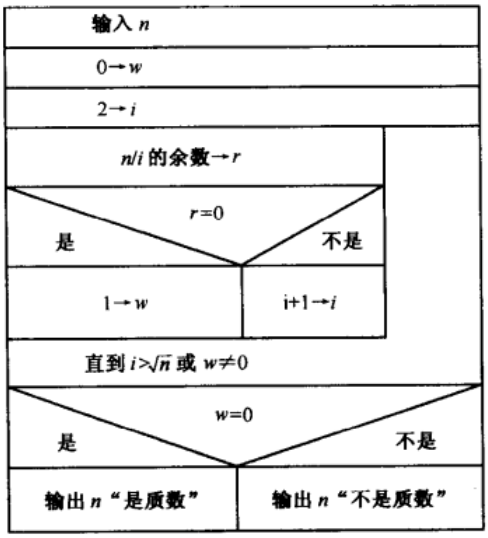


图 7-13 例 7-5 的 N-S 流程图

通过以上的例子可以看出，用 N-S 流程图表示算法的优势，即比文字描述直观、形象、易于理解；比传统流程图紧凑易画，尤其是它废除了流程线，整个算法结构是由各个基本结构按顺序组成的。N-S 流程图中的上下顺序就是执行时的顺序，即图中位置在上面的先执行，位置在下面的后执行。写算法和看算法只需从上到下进行就可以了，十分方便。用 N-S 流程图表示的算法都是结构化的算法（它不可能出现流程无规律的跳转，而只能自上而下地顺序执行）。

N-S 流程图如同一个多层的盒子，因此又将其称为盒图。

7.3.4 伪代码描述算法

用传统的流程图和 N-S 流程图描述算法直观易懂，但画起来比较费事。在设计一个算法时，可能要反复修改，而修改流程图是比较麻烦的。因此，流程图适合于表示一个算法，但在设计算法过程中使用则不是很理想（尤其是当算法比较复杂，需要反复修改时）。为此，常用一种被称为伪代码（pseudo code）的工具，以便于算法的设计。

伪代码使用介于自然语言和计算机语言之间的文字和符号来描述算法，如同一篇文章，自上而下地写下来，每一行（或几行）表示一个基本操作。它不用图形符号，因此书写方便，格式紧凑，也比较好懂，便于向计算机语言算法（即程序）过渡。

例如，“打印 x 的绝对值”的算法可以用伪代码表示如下：

```
IF x is positive THEN
    PRINT x
ELSE
  PRINT －x
```

它像一个英语句子一样好懂，在国外使用得比较普遍，也可以用汉字伪代码，如

```
若 x 为正
  打印 x
```

否则

```
    打印 －x
```

也可以中、英文混用，如

```
if x 为正
    print x
```

否则

```
print －x
```

计算机语言中具有的语句关键字用英文表示，其他的可用汉字表示，以便于书写和阅读。用伪代码写算法并无固定的、严格的语法规则，只要把意思表达清楚、书写格式清晰易读即可。

从以上例子可以看到，伪代码书写格式比较自由，可以随手写下去，容易表达出设计者的思想；同时，用伪代码写的算法很容易修改，如加一行、删一行或将后面某一部分调到前面某一位置，都是很容易做到的，而这却是用流程图表示算法时所不便处理的。用伪代码很容易写出结构化的算法。但是，用伪代码写算法不如流程图直观，可能会出现逻辑上的错误（如将循环或选择结构的范围搞错等）。

7.3.5　用计算机语言描述算法

完成一件工作应包括设计算法和实现算法两个部分。例如，作曲家创作一首曲谱就是设计一个算法，但它仅仅是一个曲谱，并未变成音乐，而作曲家的目的是希望人们能听到悦耳的音乐。由演奏家按照乐谱的规定进行演奏，这是“实现算法”。在没有人实现它时，乐谱是不会自动发声的。一个菜谱是一个算法，厨师炒菜就是实现这个算法，故需要考虑如何实现一个算法。

至此，本书只是在描述算法，并用不同的形式表示操作的步骤，而要得到运算结果，就必须实现算法。

在例 7-1、例 7-4 中用不同的形式描述了求 10！的算法，但是并没有真正求出 10！的值。实现算法的方式可能不止一种。例如，对例 7-1 表示的算法可以用人工心算的方式实现而得到结果，也可以用笔算或算盘、计算器求出结果，这就是实现算法。

要用计算机解题，就是要用计算机实现算法。计算机是无法识别流程图和伪代码的。只有用计算机语言编写的程序才能被计算机执行（当然还要编译成目标程序才能被计算机识别和执行）。因此，在用流程图或伪代码描述出一个算法后，还要将它转换成计算机语言程序。

用计算机语言表示算法必须严格遵循所用编程语言的语法规则。它不同于伪代码。下面将前面介绍过的算法用 C 语言实现。

[例 7-6]　将例 7-1 中求 10！的算法用 C 语言实现。

其程序如下：

```
main()
{
    int i,t;
```

```
    t=1;
    i=2;
    while(i<=10)
      {t=t*i;
        i=i+1;
      }
    printf("%d",t);
}
```

在这里，不打算详细介绍以上程序的细节，读者只需大体看懂即可。在以后各章节中将详细介绍有关的C语言语法规则。

应当强调说明，写出了C语言程序，仍然只是描述了算法，并非实现算法，只有运行程序才是实现算法。应该说，用计算机语言表示的算法是计算机能够执行的算法。

小结

算法设计是程序设计的关键。程序只不过是算法的一种具体表现形式而已。因此，算法是程序设计的基础。

本章内容包括算法基础知识、简单算法举例和算法的表示三部分内容。

在算法基础知识中，主要介绍的是算法的概念和算法的特点，可让学生对算法有一个基本的认识。

在简单算法举例中，主要介绍的是程序设计中常用的算法，为后续章节的程序设计奠定基础。

在算法的表示中，主要介绍的是用自然语言、传统流程图、N-S流程图、伪代码和计算机语言等几种表示算法的方法，使学生能够看懂用传统流程图表示的算法。

习题

1. 什么是算法？试从日常生活中找出两个例子，描述它们的算法。

2. 用传统流程图表示求解以下问题的算法。

(1) 有两个瓶子A和B，分别盛放醋和酱油，要求将它们互换（即A瓶原来盛醋，现改盛酱油，B瓶则相反）。

(2) 有3个数 a，b，c，要求将其中最大的数打印出来。

(3) 求 $1+2+3+\cdots\cdots+50$ 的和。

(4) 判断一个数 n 能否被2和7整除。

(5) 求两个数 x 和 y 的最大公约数。

3. 用N-S流程图表示第1题中的算法。

4. 三种基本结构的特点是什么？你能否再另外设计一种基本结构（要符合基本结构的特点）。

5. 用伪代码表示第2题中各题的算法。

第 8 章　数据类型、运算符与表达式

本章要点

- ☑ C 语言的基本语法单位
- ☑ 常量与变量、变量初始化
- ☑ 基本数据类型
- ☑ 运算符和表达式
- ☑ 运算符的结合性和优先性

8.1　C 语言的基本语法单位

1. 字符集

字符是构成程序设计语言的最小语法单位。C 语言的基本字符集包括数字、大/小写字母和特殊符号三类。

- 数字：0，1，2，3，4，5，6，7，8，9。
- 英文字母：小写字母、大写字母。
- 特殊字符：空格，!，#，%，*，(，)，+，-，_，=，|，{，[，]，;，'，:，~，?，"，.，\。

若在程序中使用了其他字符，则编译时会告知：语法错误（syntax error）。

2. 关键字

关键字又称保留字，是程序设计语言中规定的、有固定含义的单词符号。

C 语言中的关键字有 auto，break，case，char，const，continue，default，do，double，else，enum，extern，float，for，goto，if，int，register，return，short，signed，sizeof，static，struct，switch，typedef，union，unsigned，void，volatile，while。

3. 标识符

标识符是程序设计者为自定义的变量、函数、类型等所起的名字。在 C 语言中，标识符的命名必须遵循以下规则：

- 由字母、数字和下划线组成，且不能以数字开头；
- 不能与关键字同名；
- 尽量“见名知义”，提高程序的可读性，如变量名“length”表示周长。

再如，下面都是合法的标识符：

a，p1，x2c，d_1，sum，a2，IF，stu_name，total，area，day，month

而以下标识符均为不合法：

2b，x+y，g$，printf，d-2，s.day，int，a>b，#num

8.2 常量与变量

程序中要处理的数据有常量与变量之分。常量是在程序执行期间其值不能被改变的量，即常数。变量是在程序运行期间其值可以改变的量。

8.2.1 常量

1. 直接常量

常量可以是任意合法的类型。其类型一般从其字面形式即可判断，如2、3、-5等是整型常量，2.4、3.8是实型常量，'a'、'b'、'8'是字符型常量，"ABC"是字符串常量。因此，常量在程序中不必进行任何说明就可以直接使用。这种常量被称为直接常量。

2. 符号常量

符号常量是一类特殊的常量，即用一个标识符代表一个常量。其值在程序的运行过程中也是不能改变的。如程序中用#define命令定义标识符PI，代表常量3.141 592 6。在程序编译时，首先进行预处理，将在程序中出现的每一个PI都用3.141 592 6替换。这与在程序中直接使用3.141 592 6效果相同。那么为什么要使用符号常量呢？其好处有如下两点。

- 提高程序的可读性。如在程序中看见PI，就知道它代表圆周率3.141 592 6，使用EOF（代表-1）表示文件结束比直接用-1更容易读懂程序。
- 减少输入工作量，且便于修改。若程序中多次使用一个常量，特别是位数较多的常量，反复输入很麻烦，而且容易出错，而使用符号常量，则可以实现"一改全改"。例如，若程序中圆周率的值需精确到小数点后两位，则只需将第一行的符号常量更改为"3.14"即可，而不必在程序中使用该常量的地方逐一修改。

特别注意：

- 符号常量名一般用大写，而变量名一般用小写，以示区别。
- 符号常量必须先定义，后使用，如程序中的PI。

8.2.2 变量

变量是用来保存数据的。每一个变量都有一个名字，代表内存中指定的存储单元。该单元中的数据就是变量的值。变量名是一个标识符。程序通过变量名访问变量的值，如图8-1所示。x是变量名，矩形框表示某个存储单元，单元中的数据5即为变量x的值。



图8-1 变量示意图

那么变量所对应的存储单元的大小是如何确定的呢？是由变量的类型决定的。例如，在程序中有如下说明：

```
int a,b;
```

则语句的作用有两个：

- 在程序运行时，系统根据变量的类型为其分配大小合适的存储单元，如系统根据a、b的类型int为a、b各分配两个字节，并按整数的格式存放数据；

- 编译系统根据变量的类型进行语法检查，即检查对该变量的操作是否合法，如已说明 a、b 为整型变量，则 a%b（a 除以 b 取余数）是合法的；如说明“float a，b;”，则 a%b 是不允许的运算，编译时提示“非法使用浮点数”的错误信息。

基于以上原因，所以“变量必须先定义，后使用”。若使用没有定义的变量名，则编译时会提示“变量未曾定义”的错误信息。

8.3　基本数据类型

数据类型决定数据的存储形式、取值范围和允许进行的操作。因此，在学习数据类型时，主要应掌握每种类型常量的表示、定义变量的格式、所占存储空间的大小及取值范围等，从而学会根据需要处理数据确定变量的类型。

8.3.1　整型数据

1. 整型常量的表示

整型常量即整常数，可以用以下 3 种形式表示。

(1) 十进制整数，如 123、+7、-234、78 等。

(2) 八进制整数，以 0 开头的整型常量，如 012、-045、065 等分别表示 $(12)_8$、$(-45)_8$、$(65)_8$。

(3) 十六进制整数，以 0x 开头的整型常量，如 0x234、0x256 分别表示 $(234)_{16}$、$(256)_{16}$。

说明：长整型常量的末尾以字符“L”结束，如 145L、34L 是长整型常量。在 Turbo C 中，10 与 10L 的区别是：前者是基本整型常量，占两个字节的存储空间；后者是长整型常量，占 4 个字节的空间。

2. 整型变量

整型变量是用来存放整型数据的。

(1) 整型变量的分类

根据数据在内存中占据的存储单元的字节数，整型变量又可以细分为以下几种类型。

有符号型：数据所占的存储单元的最高位是符号位，其余二进制位是数值位。

有符号的整型变量有三类：基本整型——int；短整型——short [int]；长整型——long [int]。[] 表示此项可选，如 long int k 与 long k 等价。

无符号型：变量所占的存储单元的全部二进制位都是数值位，没有符号位，相当于正整数。无符号的整型变量也有三类：无符号基本整型——unsigned int；无符号短整型——unsigned short [int]；无符号长整型——unsigned long [int]。

(2) 各种类型变量所占的字节数及取值范围

标准 C 没有具体规定各类数据所占的内存字节数，一般只要求 long 型数据的长度不短于 int 型，int 型数据的长度不短于 short 型，通常随机器而异。如在微机上，int 型变量占两个字节（16 位），long 型变量占 4 个字节（32 位），而在 VAX 750 机上，int 和 long 型都占 4 个字节。以 PC 机为例，在表 8-1 中列出了 Turbo C 中规定的各类整型数据的位数及取值范围。

表 8-1 各类整型数据的位数及取值范围

数据类型	占用字节数	取值范围
int	2（16 位）	-32 768～32 767
short	2（16 位）	-32 768～32 767
long	4（32 位）	-2 147 483 648～2 147 483 647
unsigned int	2（16 位）	0～65 535
unsigned short	2（16 位）	0～65 535
unsigned long	4（32 位）	0～4 294 967 295

在实际应用中，最常用的是 int 和 long 类型，因此要熟记这两种类型的长度和范围。

(3) 整型变量的定义

变量必须先定义，后使用。因此，程序中必须先说明要使用的各变量的类型。例如，

```
int a;                    (说明变量 a 为整型)
long x,y;                 (定义变量 x,y 为长整型变量)
unsigned w;               (说明变量 w 为无符号整型)
```

可见，变量定义的一般格式为：

```
类型说明符  变量表列;
```

其中，变量表列中的多个变量之间用逗号“,”分开，变量说明语句最后以分号“;”结束。

[例 8-1] 设程序中要处理的数据有两个：1 024 和 65 536，分别用变量 x 和 y 保存，问如何确定 x 和 y 的类型？

首先明确程序中要处理的数据为整型，由表 8-1 可知，不同类型的变量所能存放的数据的范围不同。1 024 在 int 型数据的范围内，因此 x 可定义为整型，也可以定义为长整型；而 65 536 已超出了 int 型变量的表示范围，因此变量 y 必须定义为 long 型。其程序如下：

```
main()
{int x;long y;                    /*定义整型变量 x,长整型变量 y*/
  x=1024;y=65536;                 /*给变量 x、y 赋值*/
  printf("\nx=%d,y=%ld",x,y);     /*输出的值*/
}
```

其运行结果为：

```
x=1024,y=65536
```

在程序运行时，分别给 x，y 变量分配两个字节、4 个字节的存储单元，执行赋值语句后，将常量 1 024、65 536 分别送入 x，y 对应的单元，具体存储情况如图 8-2 所示。最后用 printf 函数输出变量 x，y 的值。

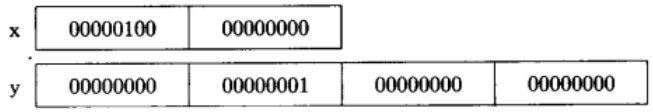


图 8-2 变量 x、y 内存中示意图

如果 x，y 均定义为 long 型变量，则程序也完全正确，但浪费存储空间，降低运行效率。

如果 x，y 均定义为 int 型变量，同时“%ld”改为“%d”，则程序运行时无错误信息提示，但结果为：

$$x=1024,\ y=0$$

很明显，变量的输出结果是错误的。因为，int 型变量能保存的最大值是 32 767，将 65 536赋给 y 时产生了“溢出”，取 65 536 存储时的低 16 位二进制值送给 y。

8.3.2　实型数据

1. 实型常量的表示

实型常量即实数，又称浮点数。它有两种表示形式。

(1) 十进制小数形式

在 C 语言中，实数只能用十进制表示，不能用八进制和十六进制表示。实数由正/负号、数字和小数点组成，而且必须有小数点。例如，0.34、3.4、7.89 等都是实型常量的小数形式。

(2) 指数形式

所谓指数形式，是指用科学计数法表示的实型常量，如 12.5e（E）3 表示实数 12.5×10^3，33e（E）－3 表示实数 33×10^{-3}等。其中，e（E）前面的数被称为尾数，e（E）后面的数被称为指数，而且指数必须是整数。在指数形式的实型常量中，尾数、e（E）、指数三者缺一不可，如 e4、5e2.3 和 3.5e 等都是错误的指数形式。

2. 实型变量

实型变量是用来保存实型数据的。

(1) 实型变量的分类

在 C 语言中，实型变量分为三类：单精度实型——float；双精度实型——double；长双精度实型——long double。

(2) 各种实型变量所占的字节数、有效数字及取值范围

标准 C 中并不具体规定每种实型数据的长度、精度和数值范围，通常因机器系统而异。表 8-2 列出的是计算机上常用的 C 编译系统的情况，仅供参考。

表 8-2　计算机上常用的 C 编译系统的情况

类　型	所占字节数	有效数字	取 值 范 围
float	4（32 位）	6～7	$10^{-37}\sim10^{38}$
duoble	8（64 位）	15～16	$10^{-307}\sim10^{308}$
long double	10（80 位）	18～19	$10^{-4931}\sim10^{4932}$

在实际应用中，最常用的是 float 和 double 两种类型，但必须准确掌握二者的不同之处，区别使用。

(3) 实型变量的定义

每一个变量使用之前必须先定义。实型变量的定义格式同整型变量，如

```
float a,b,c;                    /*定义三个单精度实型变量 a、b、c*/
double x,y;                     /*定义两双精度实型变量 x、y*/
```

编程时，如何确定变量的具体类型呢?

[例 8-2] 某程序中要处理的数据有两个：123.5 和 123.456789e10，设分别用变量 m、n 保存，确定 m、n 的类型并编程验证。

程序中要处理的这两个数据均为实型数据。根据表 8-2 可知，不同的实型变量所能保存的数据的范围和精度不同。变量 m 保存 123.5，用 float 型可以满足其范围和精度要求，因此变量 m 可定义为 float 型；变量 n 要保存的数据的有效数字为 9 位，而 float 型最多有 7 位，因此变量 n 应定义为 double 型。其程序如下：

```
main()
{ float m;double n;                    /*定义单精度型变量 m,双精度型变量 n*/
  m=123.5;n=123.456789e10              /*分别给变量 m,n 赋值*/
  printf("\nm=%f,n=%f",m,n);           /*输出 m,n 的值*/
}
```

运行结果为：

m=123.500000，n=1234567890000.000000

从运行结果看，变量的类型满足题目要求。

若将变量 n 定义为 float 型，则运行结果为

m=123.500000，n=1234567954432.000000

可以看到，m 的结果是正确的，而 n 的值从第 8 位起已经改变，这是因为 float 型数据的有效数字最多只有 7 位，超出的部分是没有意义的，所以不能定义为 float 型。

注意：编程时，根据变量要保存的数据确定变量的数据类型。如果是整型数据，则根据数据的取值范围确定其具体类型，一般较小的数据使用 int 型，较大的数据使用 long 型；如果是实型数据，则根据数据的取值范围和精度两个方面选择合适的类型。一般数据很大或精度要求较高时选用 double 型，数据较小且精度要求不高时使用 float 型。

8.3.3 字符型数据

计算机要处理的数据可以分为两大类，即数值数据和符号数据。如学生成绩、工资、价格等表示大小的是数值数据，通常用整型和实型表示；学号、姓名、住址等是符号类数据，通常用字符型表示。

1. 字符常量

C 语言中的字符常量有两种：

- 用单引号括起来的一个字符，如 'B'，'8'，'#' 等都是字符常量；
- 用单引号括起来的转义字符。转义字符是一种特殊的字符常量，是以下“\”开头的字符序列，“\”后的字符已经转变成另外的意义。例如，前面在函数中的“\n”表示一个换行符。表 8-3 列出了常用的转义字符及其含义。

表 8-3　常用的转义字符及其含义

字符形式	ASCII 码值	含　义
\0	0	空操作
\f	12	换页
\t	9	横向跳格
\b	8	退格
\a	7	报警
\\	92	反斜杠字符
\'	39	单引号字符
\"	34	双引号字符
\ddd		ASCII 码为 1～3 位 8 进制数 ddd 的字符
\xhh		ASCII 码为 1－2 位 16 进制数 hh 的字符
\r	13	回车，将当前位置移至本行开头
\n	10	换行，当前位置移至下一行开头

[例 8-3]　转义字符的使用。

```
main()
{ printf("\na\tb\101");
  printf("\nk\"\x42");
}
```

其运行结果为：

```
a       bA
k" B
```

在程序中，用 printf（）函数直接输出双引号内的各个字符。第一个 printf（）函数先输出“\n”，使光标移到新的一行的开头，接着输出字符“a”，然后输出“\t”，光标跳到下个制表位，接着输出字符“b”，再输出 ASCII 码为八进制数 101（十进制数为 65）的字符“A”。第二个 printf（）函数输出“\n”换行，然后输出字符“k”，接着输出字符双引号，再输出空格字符，然后输出 ASCII 码为十六进制数 42（十进制数为 66）的字符“B”。

2. 字符串常量

字符串常量是用双引号括起来的一串字符序列。例如，“X”，“WangLin”等都是字符串常量。其中，双引号是定界符，它不是字符串的有效字符。不含有效字符的字符串“”称为空串。

注意：注意区分字符常量与字符串常量，字符串常量存储时，自动在字符串尾部加上“\0”作为字符串结束标志。

例如，“a”与‘a’在存储时的区别如下：字符串常量“a”占两个字节，而‘a’占 1 个字节，如图 8-3 所示。实际上，在内存中存放的是字符的 ASCII 码，即字符‘a’的 ASCII 码 $(97)_{10}$，字符‘\0’的 ASCII 码（0）。



图 8-3 字符与字符串在内存中的区别

3. 字符变量

字符变量是用来存放字符常量的。字符类型说明符是 char，一个字符型变量占一个字节的存储空间，只能存放一个字符。

[例 8-4] 定义两个字符变量 c、d，用于保存字符 'a'，'b'，然后按字符和数值两种形式输出 c、d 的值。其程序如下：

```
main()
{ char c,d                                  /* 定义字符变量 c、d */
  c='a';d='b';                              /* 将字符常量'a'、'b'分别赋给变量 c、d */
  printf("\n%d    %d",c,d);                 /* 输出变量 c、d 中字符的 ASCII 码值 */
  printf("\n%c    %c",c,d);                 /* 输出变量 c、d 中的字符 */
}
```

其运行结果为：

```
a    b
97   98
```

在程序中，c、d 的存储情况如图 8-4 所示。其中，十进制数 97（即二进制数 01100001）是字符 'a' 的 ASCII 码。



图 8-4 [例 8-4] 的运行结果

可以看出，字符变量中存放的是一个字符的 ASCII 码值（二进制形式的整数）。

若将过程的第三行改为

```
c=97;d=98;
```

则程序的运行结果不变。因为执行 "c= 'a';" 时，实际上是将字符 'a' 的 ASCII 码值 97 送给了变量 c，与 "c=97;" 等价。

[例 8-5] 对字符数据进行算术运算，实现将小写字母转换成大写字母并求出下一个字母。

其程序如下：

```
main()
  { char c1,c2;                 /* 定义字符型变量 c1、c2 */
    c1='a';                     /* 将字母'a'赋给变量 c1 */
    c1=c1-32;                   /* 小写字母的 ASCII 码比大写字母的 ASCII 码大 32 */
   c2=c1+1;                     /* 相邻字符的 ASCII 码差 1 */
   printf("\n%c  %c",c1,c2);
```

```
    printf("\n%d  %d",c1,c2);
}
```

其运行结果为：

```
A   B
65  66
```

特别提示： 字符型数据与整型数据在 ASCII 码范围（0～255）内是通用的。

C 语言中没有字符串变量。编程时，字符串是用字符数组来保存的。

8.4　变量的初始化

变量初始化是指在定义变量的同时给变量赋初值，而且初值必须是确定的，如

```
int   a=23;                          /*定义 a 为整型变量,初值为 23*/
float  x=11.3;                       /*定义 x 为实型变量,初值为 11.3*/
char  c='y';                         /*定义 c 为字符型变量,初值为'y'*/
int   d1=7,d2=9;                     /*定义 d1、d2 为整型变量,初值分别为 7、9*/
char s1='2',s2='r',ss=s1;            /*多个变量初始化*/
```

特别提示： 若对几个变量赋同一个初值，则也必须分别初始化。

例如，下面语句定义 3 个变量 i、j、k 赋初值 0，应写为

```
int  i=0,j=0,k=0;
```

而不能写成

```
int  i=j=k=0;
```

变量初始化的应用：若在定义变量时就已知该变量在程序开始运行时的值，则可以用变量初始化的方式给变量赋值。没有初始化的变量，其值一般是一个随机数。

8.5　运算符和表达式

C 语言提供了丰富的运算符，这样就可以对数据进行各种处理，从而保证了各种操作的方便实现。但是在 C 语言提供的运算符中，有些不同功能的运算符使用了相同的符号。另外，运算符还具有优先级和结合性等特性。所以在学习运算符时，要掌握每一个运算符的功能、优先级和结合性，以及使用时的格式规定。

表达式是由运算符和运算对象组成的式子。运算对象就是在程序中要处理的各种数据。最简单的表达式就是一个常量或变量。在表达式中可以使用圆括号来改变优先级，任何一个表达式经过计算机都应有一个确定的值和类型。

8.5.1　算术运算符和算术表达式

1. 基本算术运算符

基本算术运算符包括 +（加），-（减），*（乘），/（除）和 %（求余数）。

这5个运算符的优先级为：“*”、“/”和“%”同级，但比“+”和“-”高，即先乘除后加减。它们对整型、单精度实型、双精度实型变量都适用，而“%”运算符只用于整型运算。另外两个整数相除，结果为一整数；若分子小于分母，则结果为0。

这5个运算符的结合性为自左至右。例如：

$$10+6-4*2$$

第一步先计算10+6，得结果为16；第二步计算4*2，得结果为8；然后用第一步计算的结果16减去第二步计算的结果8，得结果为8。

2. 算术表达式

算术表达式是由算术运算符和操作数组成的表达式。表达式的值是一个数值。表达式的类型具体由运算符和操作数确定。一般表达式中的操作类型相同时，表达式的类型就是操作数的类型。若表达式中的操作数类型不相同时，则表达式的类型是各操作数中类型最高的类型。

例如，“5+3*（6-2)”表达式的类型为整型；“3+4.0-3/2.0”表达式的类型为双精度型。

3. 数据类型转换

C语言允许不同类型的操作数据进行混合运算，但运算时系统先将操作数转换成同一类型数据，然后进行运算。一般运算符两边的操作数类型要相同，如果操作数类型不相同，则转换为高的类型。类型转换分为两种：一种是隐含转换；另一种是强制转换。

隐含转换的转换规则如图8-5所示。

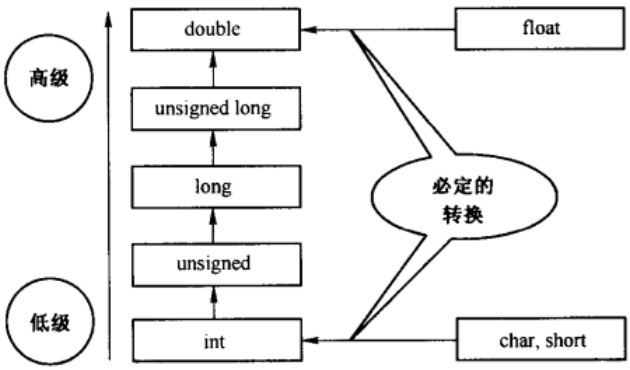


图8-5 隐含转换的转换规则

这里，整型最低，双精度最高。short和char型自动转换成int型，单精度型自动转换成双精度型。这种隐含的类型转换是一种保值映射，即在转换中数据的精度不受损失。

例如，若有int i；float f；double d；long e，则8+‘a’+i*f-d/e表达式的运算次序为：

(1) 将‘a’转换成97，然后进行8+‘a’运算；

(2) 将i和f都转换成double型，然后进行i*f运算；

(3) 将步骤（1）的结果转换为双精度型，然后与步骤（2）的结果相加；

(4) 将e转换成双精度型，然后进行d/e运算；

(5) 用步骤（3）的结果减去步骤（4）的结果。

强制类型转换是将某种类型强制性地转换为指定的类型。其格式为

（类型名）（表达式）

若有 int a；float x，y;，则（double）a 是将 int 型变量 a 的值强制性地转换为一个double型的临时中间数值，而变量 a 的类型仍为 int 型。（int）（x＋y）是将两个单精度变量 x 加 y 的结果强制地转换为一个 int 型的临时中间数值。

这种转换是不安全的，因为强制性地转换可能会出现将高类型向低类型转换。这时数据精度要受到损失，甚至得到一个毫无意义的数据。

例如，若 x＝2.33；y＝2.60;，则（int）（x＋y）的结果为 4。

在这个例子中，如果变量 x 和 y 的值比较大，则当它们之和的值大于整型的有效范围时，（int）（x＋y）被强制性转换结果为一个无意义的数值。

另外，强制类型转换（int）（x＋y）仅将 x＋y 的结果强制性转换为整型，而并不能改变变量 x 和 y 的原单精度类型，即变量 x 和 y 的类型仍为单精度型。

8.5.2　赋值运算符和赋值表达式

1. 赋值运算符

赋值运算符是一种能够改变变量值的运算符，即它在计算赋值表达式的值后，还将会改变其变量的值。在 C 语言中，赋值运算符分为简单的赋值运算符和复合的赋值运算符。

简单赋值运算符只有一个，即＝（赋值运算符）。

将赋值运算符右边表达式的值赋给赋值运算符左边的一个变量。

例如：

```
int x＝6;
x＝5＋10;
```

第一个语句定义了一个整型的变量 x，并且赋初值 6。第二个语句是将赋值运算符右边的表达式 5＋10 计算，将结果 15 赋给赋值运算符左边的 x 变量，从而使变量 x 的值由原来的 6 改变为 15。

复合赋值运算符共有 10 个，即＋＝（加赋值）、－＝（减赋值）、＊＝（乘赋值）、/＝（除赋值）、％＝（求余数赋值）、&＝（按位与赋值）、|＝（按位或赋值）、^＝（按位异或赋值）、<<＝（左移位赋值）及>>＝（右移位赋值）。

复合赋值运算符是先进行一种算术运算后，再进行赋值。

例如：

```
int x＝6;
x＊＝3;
```

第一个语句定义一个整型变量 x，并且赋初值 6。第二个语句是先进行 x＊3 表达式计算，将结果 18 赋给赋值运算符左边的 x 变量，从而使变量 x 的值由原来的 6 改变为 18，即 x＊＝3 等价于 x＝x＊3。其他复合赋值运算符的含义相似。

2. 赋值表达式

由赋值运算符和操作对象连接组成的式子被称为赋值表达式。赋值运算符的结合性是从

右至左的。

例如：

```
int x,y,z;
x=y=z=5+7;
```

上面第二个语句的执行是：先计算 z=5+7 的值，计算后，z 被更新为 12。表达式的值也是 12。接着，又将表达式 z=5+7 的值 12 赋给变量 y。最后将表达式 y=z=5+7 的值 12 赋给变量 x。经过上面的连续赋值，使变量 x、y、z 的值都更新为 12。

在计算复合赋值运算符的表达式时，应该先计算右边表达式的值后再与左边变量做运算，将结果赋给复合赋值运算符左边的变量。例如：

```
int x=3,y=4;
x*=y+1;     /*等价于 x=x*(y+1);*/
```

这里，x 的值为 15。因为先计算右边表达式 y+1，然后再与变量 x 相乘，将结果赋给变量 x。

赋值运算符的数据类型若不一致，则要进行类型转换。这也是一种强制类型转换，被称为隐含类型转换。其方式为：在赋值表达式中，当左边变量（赋值运算符左边的表达式）和右边表达式（赋值运算符右边的表达式）的类型不同时，一律将右边表达式的类型强制转换为左边变量的类型。

例如：

```
int x;
float y=5.26;
x=y+1.3;
```

将 y+1.3 的值 6.56 赋给整型变量 x 时，由于 6.56 是一个单精度类型数值，舍弃小数部分，将整数部分 6 赋给整型变量 x。注意，当赋值运算符右边表达式的值大于整型的有效范围时，将被赋值一个无意义的数值。当将高类型向低类型转换时，都存在这个问题。在使用时注意，避免出现错误。

将字符数据赋给整型变量时，是将字符的 ASCII 码赋给整型变量。

例如：

```
int x;
x='a';
```

它将字符'a'的 ASCII 码值 97，赋给整变量 x。

因为在由赋值运算符组成的表达式中，数据类型若不一致，则要进行类型转换，因此正确掌握转换规则，才能编制正确的程序。来看下面的一个例子：

```
int x=5,y;
y=x+6.35;
```

在上面的表达式 y=x+6.35 中，要做两次隐含转换，先将变量 x 的值转换成单精度型，然后与 6.35 进行相加运算，最后将相加的结果值转换成整型赋给变量 y。

8.5.3　关系运算符和关系表达式

1. 关系运算符

C 语言中的关系运算符共有 6 个，即

＞（大于）、＜（小于）、＞＝（大于等于）、＜＝（小于等于）、==（等于）、！＝（不等于）。

前面 4 个的优先级高于后面的两个。

关系运算符的运算比较简单，在此不再详述。

2. 关系表达式

由关系运算符和操作数组成的表达式被称为关系表达式。关系表达式的值是一个逻辑型的值，即只有两个值（真和假）。这里要说明的是，C 语言是用 1 作为真，用 0 作为假的。但是在进行运算时，非 0 即认为真，0 才认为假。而表达式的结果为真时，给出真值 1；为假时，给出假值 0。

例如，若有 int x＝2，y＝3，z＝－5，则：x＞y，结果为 0；z＞＝y，结果为 0；z==y，结果为 0；z＞y＞x，结果为 0。

关系运算符的结合性为自左至右。

例如，x＞y！＝z＞y 结果为 0。为什么？请分析。

关系运算符的优先级低于算术运算符。

例如，y＞＝x＋x，结果为 0；x＞＝z－y，结果为 1。

8.5.4　逻辑运算符和逻辑表达式

1. 逻辑运算符

逻辑运算符有以下 3 个。

- 单目逻辑运算符：！（逻辑求反）。
- 双目逻辑运算符：&&（逻辑与）、||（逻辑或）。
- &&（逻辑与）的优先级高于||（逻辑或），而！（逻辑求反）最高。

2. 逻辑表达式

由逻辑运算符连接关系表达式或逻辑量组成的表达式被称为逻辑表达式。逻辑表达式的值也是一个逻辑值，即只有真和假值，用 1 作为真，用 0 作为假。运算时非 0 即认为真，0 才认为假。而表达式的结果为真时，给出真值 1；为假时，给出假值 0。

逻辑求反（！）：则真求反后为假，而假求反后是真。

逻辑与（&&）：两个操作数都为真时结果为真，有一个操作数为假时结果为假。

逻辑或（||）：两个操作数都为假时结果为假，有一个操作数为真时结果为真。

例如：

```
int x=6,y=3,z=3;
! x          结果为 0。
x&&y         结果为 1。
! x||y       结果为 1。
```

逻辑运算符的结合性为自左至右。逻辑与（&&）和逻辑或（||）运算符的优先级低于关系运算符。逻辑求反（!）运算符的优先级高于算术运算符。

例如：

```
int  x=4,y=5,z=9;
x>y&&z>y          结果为0。
x&&! y&&z         结果为0。
x<y||y<z          结果为1。
x<z&&y-5          结果为0。
x<z&&!(y-5)       结果为1。
```

在由“&&”和“||”运算符组成的逻辑表达式中，C语言规定，只对能够确定整个表达式值所需要的最少数目的子表达式进行计算，即在由若干个子表达式（或操作数）组成的一个逻辑表达式中，从左至右计算子表达式的值，当计算出一个子表达式的值便可确定整个逻辑表达式的值时，后面的子表达式就不再计算了。

例如：

```
int  x=3,y=0;
! x&&x+y&&x+1          结果为0。
```

当计算出！x为0时，便可确定整个逻辑表达式的值为0。于是，后面的子表达式x+y和x+1就不再计算了。

再例如：

```
int x=3,y=0;
x||y||y+x||y=1          结果为1。
```

当计算出x为3时，便可确定整个逻辑表达式的值为1。于是，后面的子表达式y+x和y=1就不再计算了。此时，y的值仍为0，而不是1。

8.5.5 自增、自减运算符

自增、自减运算符：++，--。

这两个运算符都是单目运算符，也是一种能够改变变量值的运算符，即它在计算表达式的值后，还将会改变其变量的值（使变量的值增1或减1）。

例如：

```
int x=1;
++x;
```

表达式++x的值为2，它是x+1的值，同时变量的值改变为2。因而表达式++x等价于赋值表达式x=x+1。

C语言的这两个运算符既可以用于一个变量的前面，也可以用于一个变量的后面。不论作用于一个变量的前面，还是后面，都会使被作用的变量值加1或减1。但作用于一个变量的前面，则该变量值先被加1或减1，然后该变量才被引用。反之，如作用于一个变量的后面，则先引用该变量，然后该变量值被加1或减1。

例如：

```
++x;--x;    (先自增 1、自减 1)。
x++;x--;    (后自增 1、自减 1)。
```

设有

```
int x1,x2,x3,x4,y=5;
x1= ++y;x2= --y;
x3=y++;x4=y--;
```

则经过运算，x1 的值为 6，x2 的值为 5，x3 的值为 5，x4 的值为 6。为什么？请分析。

[例 8-6]　自增、自减运算符应用举例。

```
main()
{ int x=9,y,z;
  y=(++x)+(x++)+(++x);
  z=(--x)+(x--)+(--x);
  printf("y=%d,z=%d,x=%d",y,z,x);
}
```

则经过运算，y 的值为 30，z 的值为 33，而变量 x 的值为 10。为什么？这是因为当在表达式中含有自增自减运算时，C 语言系统一般按以下三步来完成计算：

- 将所有先自增、自减运算抽出来进行计算；
- 将第 1 步计算的结果带入表达式中，计算表达式的值；
- 再将所有后自增、自减运算抽出来进行计算。

[例 8-6] 中的表达式 y=（++x）+（x++）+（++x），C 语言系统进行两次的自增计算，使 x 的值由 8 变为 10。然后将 10 带入表达式中计算（即计算“x+x+x”），得结果为 30，并赋给变量 y，最后再进行 1 次 x 的后自增计算，使得变量 x 的值由 10 变为 11。

例 8-6 中表达式 z=（--x）+（x--）+（--x）的计算方法分析同上。请分析。

但特别要注意的是，当自增、自减运算出现在函数的参数中时，它们不按该方法进行计算。在函数中的计算方法由 C 语言系统采用扫描格式决定。

例如：

```
printf("y=%d",(++x)+(x++)+(++x));
```

结果为：y=29。即 y 的结果为 29 而不是 30。也就是说，printf（）函数不按上述方法计算。

因此，一般在进行 C 语言程序设计时，函数的参数中都避免使用自增、自减运算符（可以将函数参数中含有自增、自减运算的表达式提到函数调用的外面），从而使设计出的程序可在各种 C 语言编译系统下获得同样的计算结果。

例如，若有如下语句：

```
printf("z=%d",(++x)+(x++));
```

则可以将它改为以下格式：

```
y=(++x)+(x++);
printf("z=%d",y);
```

自增、自减运算符的结合性为自右至左。

例如：

```
-i++        相当于-(i++)
i+++j       相当于(i++)+j
```

这里要特别说明的是，像 i+++j 或 i---j 这样的表达式，大多数 C 编译系统均将它们解释为（i++）+j 和（i--）-j，但 C 语言也没有对其规定，因而有些 C 编译系统将它们解释为 i+(++j) 和 i-(--j)。在这里建议：避免使用这种表达式，可以通过加圆括号的方法，清楚地表明要进行的运算，从而使编制出的程序具有好的可移植性。

8.6 位操作运算符及表达式

位操作运算符是用来进行二进制位运算的运算符。它又分为两类，即逻辑位运算符和移位运算符。由位操作运算符连接的操作对象组成的表达式也被称为算术表达式。因为位操作运算符实质是算术运算符。它组成的表达式的值是算术值。另外，位操作只针对基本数据类型中的 char 和 int 型数据中的二进制位进行操作，不能对其他类型的数据操作。

1. 逻辑位运算符

- 单目逻辑位运算符：～（按位求反）。
- 双目逻辑位运算符：&（按位与）、|（按位或）和^（按位异或）。

在双目逻辑位运算符中，& 高于^，而^又高于|。

按位求反（～）是将各个二进制位中的 1 变为 0，0 变为 1。

按位与（&）是将两个二进制操作数从低位（最右位）到高位依次对齐后，如果两个相应的二进制位都为 1，则该位的结果为 1；否则，为 0。

[例 8-7] 127&3 结果为 3。其运算过程为：

```
    01111111    (127 的二进制)
&   00000011    (3 的二进制)
-------------
    00000011    (按位与的结果)
```

按位或（|）是将两个二进制位的操作数从低位（最右位）到高位依次对齐后，每位求或的结果是：只要有 1 位为 1 时就为 1；否则，为 0。

[例 8-8] 127|3 结果为 127。其运算过程为：

```
    01111111    (127 的二进制)
|   00000011    (3 的二进制)
-------------
    01111111    (按位或的结果)
```

按位异或（^）是将两个二进制位的操作数从低位到高位依次对齐后，每位求异或的结果是只有两位不同时就为 1；否则，为 0。

[例 8-9] 127^7 结果为 120。其运算过程为：

```
    01111111    (127 的二进制)
^   00000111    (7 的二进制)
-------------
    01111000    (按位异或的结果)
```

2. 移位运算符

两个移位运算符都是双目运算符。它们是<<（左移）和>>（右移）。

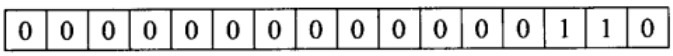
左移是将 1 个二进制的数按指定移动的位数向左移位，移掉的位被丢弃，右边移出的空位一律补 0。

[例 8-10]　使用 unsigned 型数据，左移运算符应用举例。

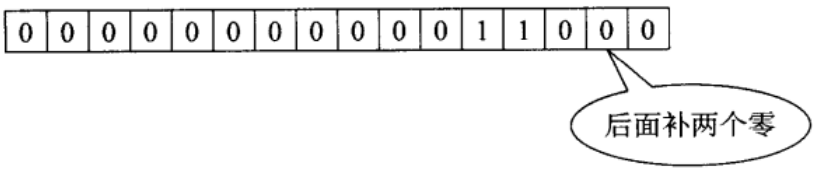
```
main()
{unsigned x=6;
x=x<<2;
printf("x=%u",x);
}
```

运行结果为 x=24。

即 x 的原值为：



左移两位后为：



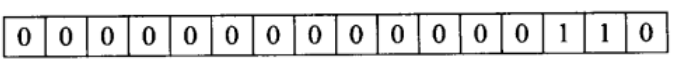
右移是将 1 个二进制的数据按移动的位数向右移位，移掉的位被丢弃，左边移出的空位或者补 0，或者补符号位，这要由机器而定。在使用补码做机器数的机器中，正数的符号位为 0，负数的符号位为 1。

[例 8-11]　使用 unsigned 型数据，则 C 编译系统一般采用逻辑右移，左边移出的空位补 0。

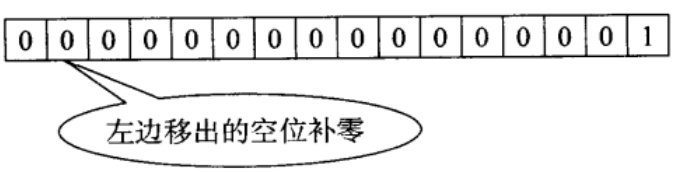
```
main()
{unsigned x=6;
x=x>>2;
printf("x=%u",x);
}
```

运行结果为 x=1。

即 x 的原值为：



右移两位后为：



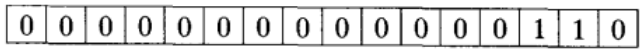
[例 8-12]　使用带符号型数据，C 编译系统一般采用算术右移，左边移出的空位用符号位填充。

```
main()
```

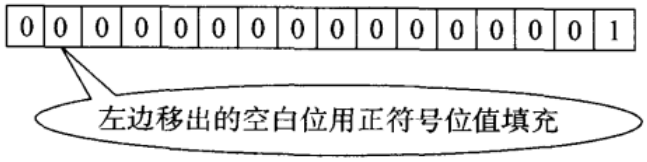
```
{int x=6;
x=x>>2;
printf("x=%d",x);}
```

运行结果为 x=1。

即 x 的原值为：



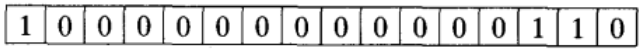
右移两位后为：



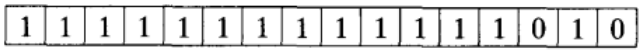

```
main()
{ int x=-6;
  x=x>>2;
  printf("x=%d",x);
}
```

运行结果为 x=-2。

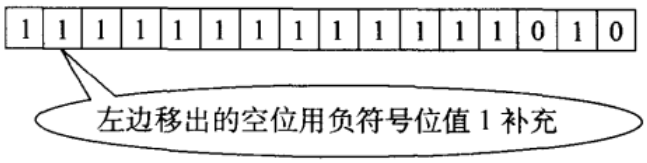
即 x 的原值为：



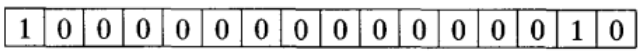
由于计算机中数据采用补码形式，故补码形式为：



右移两位后为：



将其补码形式转换成原码值形式为（即-2）：



8.7 条件运算符和条件表达式

条件运算符是 C 语言提供的一个仅有的三目运算符。该运算符需要 3 个操作数。它的格式如下：

```
d1? d2:d3
```

其中，d1、d2、d3 是 3 个操作数。

三目运算符的功能是先计算 d1 的值，再根据该值进行判断。如果为非 0，则计算 d2 的值，并把该值作为条件运算符的值；否则，计算 d3 的值，并把该值作为条件运算符的值。

而条件运算符的类型为 d2 和 d3 中类型高的一个类型。另外三目运算符结合性是自右至左的。

[例 8-13]　条件运算符应用例子。

```
main()
{int x=3,y=4,z;
z=x>y? ++x:++y;
printf("%d,%d,/%d",x,y,z);
z=x-y? x+y:x-3? x:y);
printf("%d,%d,/%d",x,y,z);
}
```

程序的运行结果为：

```
3,5,5,
3,5,8
```

8.8　逗号运算符和逗号表达式

C 语言把逗号“,”也作为运算符。逗号运算符的功能是将多个表达式组成一个逗号表达式。逗号运算符的优先级是所有运算符中最低的一个运算符。逗号表达式运算规则是：从左至右依次计算各个表达式的值，最后一个计算的表达式的值和类型便是整个逗号表达式的值和类型。

例如，$y=(x=3,\ 5+6,\ x+5)$ 逗号表达式的值为 8。

[例 8-14]　逗号运算符应用例子。

```
main()
{ int x=1,y=2,z;
  z=x+y+3;
  printf("%d,%d,%d",x,y,z);
  z=(x++,x+=y,x+y);
  printf("%d,%d,%d",x,y,z);
}
```

程序的运行结果为：

```
1,2,6
4,2,6
```

8.9　运算符的结合性和优先级

每一种运算符都有一个优先级。优先级是用来标志运算符在表达式中的运算顺序的。优先级高的先做运算；优先级低的后做运算；优先级相同的由结合性决定计算顺序。C 语言将运算符分为 15 种优先级。当然可以使用圆括号改变运算的顺序。结合性也是标志运算符在表达式中的运算顺序的。当优先级相同时，表达式的计算顺序便由结合性来决定。大多数运

算符的结合性是从左至右的。只有3类运算符的结合性是从右至左的。它们是单目、三目和赋值运算符。表8-4列出了常用运算符的优先级和结合性。

表8-4 常用运算符的优先级和结合性

优 先 级	运 算 符	结 合 性
1	()	从左至右
2	!、++、--、&、*、+、-（类型）	从右至左
3	*、/、%	从左至右
4	+、-	
5	<<、>>	
6	<、<=、>、>=	
7	==、!=	
8	&（按位与）	
9	^（按位异或）	
10	\|（按位或）	
11	&&	
12	\|\|	
13	?:	从右至左
14	=、+=、-=、*=、/=、%=、&=、^=、\|=、<<=、>>=	从右至左
15	,	从左至右

小结

数据类型、运算符和表达式是C语言中的最基本的知识，只有准确掌握才能编制出正确、符合C语法规定的语句。

本章首先介绍了数据类型的基本概念，然后详细介绍了整型、实型和字符型等三种基本类型、常用运算符（算术运算、赋值运算、关系运算、逻辑运算和位运算）及其表达式。要求重点掌握两点：① 各数据类型的常量表示、变量的定义和取值范围，强调在应用中根据变量要保存的数据的大小及精度确定变量的类型；② 各运算符的功能、优先级及结合性，表达式的正确书写及表达式的计算过程。

习题

1. 指出下面哪些是非法的标识符，并说明原因。

no.one class_1 3_year #uss romm12 int main m.i.c -student 10_am 9999 else Int integer

2. 为什么在C语言中规定变量必须先定义，后使用?

3. 字符常量与字符串常量有什么区别?

4. 假如要编程处理某商品的信息：商品代号（一个字母）、价格、数量（小于10000），设分别用变量no、price、num表示，则问：

(1) 该如何定义变量?

(2) 若已知变量要存放的数据为 'B'、89.9、2000，请分别用变量的初始化和赋值表达式语句两种方法将数据送给变量。

5. 当几种类型的数据混合运算时，各种数据类型之间进行隐式转换的规则是什么?

6. 写出以下程序运行的结果。

```
main()
{ char c1='a',c2='b',c3='c',c4='\101',c5='\115';
printf("a%b b%c\tc%c\tabc\n",c1,c2,c3);
printf("\t\b%c %c",c4,c5);
}
```

7. 计算下面表达式的值并指出结果的类型。

(1) 8/4 * (int) 2.5/(int)(1.25 * (3.7+3))

(2) 设 int a=7; float x=2, y=4.7, 计算 x+a%3 * (int)(x+y)%2/4

(3) (k=3*2, k+4), k*3

8. 将下列算术表达式写成 C 表达式。

(1) $e^2\sin\left(\frac{\pi}{4}\right)$

(2) $3x+a^2+c^4$

(3) $\frac{\sqrt{m}+n}{2}$

(4) $\frac{56y}{a+b}$

9. 设 int a, b, 且 a=10, b=5, 写出计算下面各表达式后变量 a 的值。

(1) a+=b

(2) a*=--b+4

(3) a/=a+a

(4) a%=b%=4

(5) a-=2

(6) a+=a-=a*=a

10. 什么是关系运算? 什么是逻辑运算?

11. C 语言中如何表示“真”和“假”? 系统如何判断一个量的“真”和“假”?

12. 写出下面各逻辑表达式的值。设 int x, y, a=3, b=4, c=5。

(1) a+b>c&&b==c

(2) a||b+c&&b-c

(3) !(a>b)&&!c||1

(4) !(x=a)&&(y=b)&&0

(5) !(a+b)+c-1&&b+c/2

第 9 章　结构化程序设计

本章要点

- ☑ C 语句的分类
- ☑ 常用输入输出函数
- ☑ 顺序结构程序设计
- ☑ 实现选择结构的语句：if 语句、switch 语句
- ☑ 选择结构程序设计
- ☑ 实现循环结构的语句：while 语句、for 语句、do...while
- ☑ 循环结构程序设计
- ☑ break 语句和 continue 语句

程序是由多条语句组成的，是描述计算机的执行步骤。人们利用计算机解决一个问题，必须预先将问题转化为计算机语句描述的解题步骤。也就是说，当程序在计算机上执行时，程序中的语句是用来完成具体的操作和控制计算机的执行流程的。

9.1　C 语句概述

和其他高级语言一样，C 语言的语句是用来向计算机系统发出操作指令的。一个语句经编译后产生若干条机器指令。一个实际的程序应当包含若干条语句。在 C 语言中，一个程序是由若干个函数构成的。一个函数是又由声明部分和执行部分构成的。

C 语言语句是以分号作为语句结束符的。因此，不论 1 条语句的内容有多长，只要它中间没有分号就认为是 1 条语句。C 语句可以分为以下 5 类。

(1) 表达式语句

所谓表达式语句是在表达式后边跟一个分号所构成的语句。有些表达式语句没有实际意义，但从语法上讲，它是合法的。例如：

```
i>j;
i;
1*2;
```

这三个语句无实际意义。因为它们没有保存计算结果，对程序的功能不产生任何影响。要说它们的作用，则只是延长执行时间，但它是合法的。

又如赋值语句：

```
i=j=1;
```

它的作用是给变量赋一个新值，也属赋值语句。

```
i++;
```

变量新值的大小取决于原值加“1”。

(2) 空语句

空语句是仅由一个分号表示的语句。其格式为：

```
;
```

它不产生任何作用（此语句主要用于循环体和标号所标识的语句）。

(3) 函数调用语句

当函数定义好以后，在程序中使用“函数名（参数表）”来代替整个函数调用。函数名、括号及括号内的参数表就是函数表达式。函数调用语句由函数表达式和一个分号组成。例如：

```
printf("NORTHWEST UNIVERSRTY \n");
sqrt(3);
```

虽然函数是一段程序，但是函数的调用是作为语句的形式出现的，且执行顺序是由前向后的。

(4) 复合语句

复合语句是由一对大括号括起来的若干语句组成的。复合语句中可以由说明部分对复合语句中所用到的变量进行定义。例如：

```
{int sum=0;
 sum+=i;
 i++;
}
```

复合语句是一个整体，可以存在于程序中允许语句出现的任何地方。从逻辑上讲，它是一个语句，即为单个语句。

(5) 控制语句

控制语句可完成一定的控制功能。C 语言只有 9 种控制语句。它们是：

if ()... else...	(条件语句)
for ()...	(循环语句)
while ()...	(循环语句)
do...while ()	(循环语句)
continue	(结束本次循环语句)
break	(中止执行 switch 或循环语句)
switch	(多路分支选择语句)
goto	(转向语句)
return	(从函数返回语句)

上面 9 种语句中的括号 () 表示含有一个条件，...表示内嵌的语句。例如，if () ... else ...的具体语句可以写成：

```
if(x>y) z=x;else z=y;
```

9.2 顺序结构程序设计

顺序结构程序是结构化程序设计中最简单、最常见的一种程序结构。在顺序结构程序中，程序的执行是按照各语句的先后次序顺序执行的，并且每个语句都会被执行到。

9.2.1 实现顺序结构程序设计的基本语句

1. 赋值语句

赋值语句是由赋值表达式加一个分号构成的。其最简单的一种形式为：

```
变量=表达式;
```

例如：

```
x=14.5;              /*将实型数14.5赋值给变量x*/
cir=2*3.14*r;        /*将表达式2*3.14*r的值赋值给变量cir*/
```

其中，“=”是赋值符号；赋值符号的右边是表达式；左边是变量。

因赋值语句是由赋值表达式加上一个分号构成的，所以下面也是合法的赋值语句。

```
i++;          自增赋值语句
x+=2;         复合加赋值语句
j--;          自减赋值语句
```

赋值语句是将右边表达式的值赋给左边的变量中去。因此，赋值语句要先对表达式加以计算求得其值，而后再将求得的值赋给左边的变量中去，所以赋值语句兼有计算功能。

在程序设计中，赋值语句不仅用于为变量进行赋值，同时还大量用于运算处理。

2. 标准输入输出

输入和输出在计算机程序中往往占有很大的比例，然而C语言本身并不提供输入输出语句。其输入输出操作是由标准库函数来实现的，如printf（）函数、scanf（）函数等。在使用标准库函数时，要用预编译命令：#include将stdio.h文件包括到用户源文件中。但由于printf（）和scanf（）函数使用频繁，故系统允许在使用它们时省略该预编译命令。

（1）格式输出函数

格式输出函数printf（）的功能是按指定的格式向显示器输出一个或多个任意类型的数据。printf（）函数的一般调用形式为：

```
printf("格式控制"[,输出项表]);
```

例如：

```
printf("hello");
printf("The area is :%f\n",area);
```

说明：

- 格式控制。格式控制的两边必须使用双引号，其中的内容可以包含普通字符、格式说明符和转义字符3类信息，并且这3类信息可以交叉出现。

- 普通字符：需要原样输出的字符信息，如上面语句中的“Hello”、“The area is:”就属于这一类，程序执行后，将这类信息在显示器上原样输出。
- 格式说明符：用%开头后面跟有一个字母，规定了输出项的输出形式，如上面语句中的%f，规定了输出项 area 以小数的形式输出。格式说明符在个数和类型上应与输出项相匹配。printf（）函数的格式说明符见表 9-1。

表 9-1　printf 函数的格式说明符

格式说明符	输　出
%d	带符号的十进制整数
%c	单个字符
%s	字符串
%f	以小数形式输出浮点数，隐含输出 6 位小数
%e	以指数形式输出浮点数，保留 6 位小数
%g	使用%f 和%e 中输出宽度较短者，不输出无意义的零
%o	不带正、负号的八进制整数
%u	不带正、负号的十进制整数
%x	不带正、负号的十六进制整数

[例 9-1]　输出格式举例。

```
main()
{int num1,num2;
 float real1,real2,real;
 char ch;
 num1=65;
 num2=-3;
 real1=256.34;
 real2=13.32;
 real=real1+real2;
 ch='A';
 printf("%d%c%d%o%f%c%d",num1,num1,num2,num2,real,ch,ch);
 printf("%s","Very good!");
}
```

其运行结果为：

```
65A-3177775269.660000A65Very good!
```

从例 9-1 的运行结果可以看出，整型变量可以字符的形式输出，而字符型变量也可以整数的形式输出。ASCII 码是它们之间的桥梁。

负数在计算机内存中是以补码形式存放的。变量 num2 的值是 -3。-3 的补码为 $(1111111111111101)_2$，用八进制表示即为 177775，所以变量 num2 以%o 格式输出时显示 177775。

使用格式说明符%f输出时，整数部分全部如数输出，小数部分按系统默认宽度（6位小数）输出。需要注意的是，并非全部数字都是有效数字：float型实数的有效数字是前7位，double型实数的有效数字是前16位。因此在例题中输出时，结果中只有前7位是有效的。

printf（）函数除可以使用表9-1中给出的格式说明符规定输出项的输出形式以外，还可以在%与字母（格式字符）之间插入附加格式字符（见表9-2），用于进一步修饰输出信息。

表9-2 printf（）函数附加格式说明符

字　符	说　明
字母l	用于长整型整数，可以加在格式字符d、o、x、u前面
m（代表一个正整数）	数据的输出宽度
N（代表一个正整数）	对实数，表示输出小数位的位数；对字符串，表示截取的字符个数
-	输出的数字或字符向左靠齐

[例9-2] 输出格式举例。

```
main()
{ int num1 = 123;
  long num2 = 1234567;
  float real = 123.4567;
  printf("%d, %6d, % -6d, %2d \ n", num1, num1, num1, num1);
  printf("%ld, %8ld, %4ld \ n", num2, num2, num2);
  printf("%f, %10f, %10.2f, % -10.2f \ n", real, real, real, real);
  printf("%s, %10.5s, % -10.5s \ n", "student", "student", "student");
}
```

其运行情况为：

```
123, 123, 123, 123
1234567, 1234567, 1234567
123.456703, 123.456703, 123.456703, 123.46
student, stude, stude
```

以“\”开头的字符序列，如例9-2中，printf（）函数中的“\n”就是转义字符。其作用是在输出时产生一个“换行”操作。C语言中常用的转义字符见表9-3。

表9-3 C语言中常用的转义字符

字符形式	功　能
\n	换行（光标跳到下一行首）
\t	横向跳格（即跳到下一个输出区，一个输出区占8列）
\v	竖向跳格
\b	退格

续表

字符形式	功　能
\r	回车（光标回到本行首）
\f	走纸换页
\\	反斜杠字符“\”
\'	单引号字符
\ddd	1 到 3 位八进制数所代表的字符
\xhh	1 到 2 位十六进制数所代表的字符

[例 9-3]　输出格式举例。

```
main()
{char a,b,c;
 a='n';
 b='e';
 c='\167';
 printf("%c%c%c\n",a,b,c);
 printf("%c\t%c\t%c\n",a,b,c);
 printf("%c\n%c\n%c\n",a,b,c);
}
```

其运行情况为：

```
new
n       e       w
n
e
w
```

输出项表中可以有多个输出项，各项之间用逗号分隔。

printf（）函数使用时的注意事项如下。

- printf（）函数可以输出常量、变量和表达式的值。但格式控制中的格式说明符必须按从左至右的顺序，与输出项表中的每个数据一一对应，否则将会出错。
- 类型转换字符 x，e，g 可以用小写字母，也可以用大写字母。使用大写字母时，输出数据中包含的字母也要大写。除 x，e，g 外，其他类型转换字符必须用小写字母，如%f 不能写成%F。
- 类型转换字符紧跟在“%”后面时作为类型转换字符；否则，将用为普通字符处理（即原样输出）。例如，printf（“c=%c，f=%f\n”，c，f）中的第 1 个 c 和 f 都是普通字符。

3．格式输入函数

格式输入函数 scanf（）的功能是用来从外部输入设备（通常是键盘）向程序中的变量输入一个或多个任意类型的数据。scanf（）函数的一般调用形式为：

scanf("格式控制",输入项表);

例如:

scanf("%d%f",&x,&y);

(1) 格式控制

格式控制的两边必须用双引号。其中,内容可以由格式说明符和普通字符组成。

普通字符在输入数据时,必须在对应位置上原样输入这些字符。例如:

scanf("character:%c,real:%f",&c,&f);

函数中的"character:","real:"和","都是普通字符,则在实际运行时输入:

character:w,real:12.5

scanf()函数中的格式说明符的使用与printf()函数相似,必须用"%"开头后面跟一个字母(也可以在其中间增加附加字符)。它规定了输入项对应的输入数据格式。同样,格式说明符要在个数和类型上与输入项相匹配。scanf()函数常用的格式说明符及附加字符见表9-4和表9-5。

表9-4 scanf函数常用的格式说明符

格式说明符	输 出
%d	十进制整数
%c	单个字符
%s	输入一个字符串,将字符串送到一个字符数组中,输入时以非空白字符开始,以第一个空白字符结束
%f	实数,可以用小数形式或指数形式输入
%e	同%f,e和f可以相互替换
%o	八进制整数
%x	十六进制整数

表9-5 scanf函数常用的格式符加字符

附加字符	说 明
l	用于输入长型数据%ld,%lo,%lx,%lf,%le
h	用于输入短整型数据%hd,%ho,%hx
域宽m	指定输入数据所占的列数
*	表示本输入数据在读入后不赋给相应的变量

(2) 输入项表

输入项表中是要输入数据的变量的存储单元地址。"&"是地址运算符,"& 变量名"表示变量在内存中的地址。因此,scanf()函数的作用是:按照变量c和f的内存地址将输入的字符w和实数12.5值存入其中。

(3) scanf()函数注意事项

scanf()函数在使用时,从键盘输入数据要注意以下几点。

① scanf（）函数没有计算功能，因此输入的数据只能是常量，而不能是表达式。

② 在输入多个整型数据或实型数据时，可以用一个或若干个空格、回车或制表符作为间隔。但在输入多个字符型数据时，数据之间不可以使用分隔符。例如，

```
scanf("%d%d%d",&x,&y,&z);
```

表示要输入 3 个十进制整数，可按以下方式输入：

```
12  34  56
```

或

```
12
34
56
```

例如：

```
scanf("%c%c",&c1,&c2);
```

表示要输入两个字符型数据，按以下形式输入：

```
ab
```

则字符 a 赋给变量 c1，字符 b 赋给变量 c2。

③ 在输入格式中，除格式说明符之外的普通字符应原样输入，如

```
scanf("x=%d,y=%d,z=%d",&x,&y,&z);
```

应使用以下形式输入：

```
x=12,y=34,z=67
```

④ 如果指定输入数据的宽度，则系统自动按它截取所需的数据，如

```
scanf("&4d%4d%4d",&x,&y,&z);
```

输入：

```
12345678901
```

则系统自动将 1234 赋给 x，将 5678 赋给 y，将 901 赋给 z。

注意：在输入实数时，不能规定小数位。例如：scanf（"%6.2f"，&f）；是错误的。

⑤ 在%后面使用“*”附加符，表示跳过它相应的数据，如：

```
scanf("%4d,%*4d,%4d",&x,&y);
```

输入：

```
1234,2346,9012
```

则系统将 1234 赋给 x，将 9012 赋给 y，而第 2 个数据“2346”跳过不赋给任何变量。

由于 scanf（）函数本身不具有显示提示信息功能，故为改善人－机交互性，在设计输

入操作时，一般先用 printf（）函数输出一个提示信息，然后再用 scanf（）函数进行数据输入，如：

```
printf("Enter width and height:");          /*输入提示信息*/
scanf("%f,%f",&width,&height);
```

4. 单个字符输出函数

putchar（）函数的功能是用于将一个字符输出到显示器上显示。

putchar（）函数的一般调用形式为：

```
putchar(c);
```

即把变量 c 的值输出到显示器上。这里的 c 可以是字符或整型变量，也可以是一个转义字符。

[例 9-4] putchar（）函数应用举例。

```
#include "stdio.h"
main()
{ char a,b,c,d;
  a='g';
  b='o';
  c=111;
  d='d';
  putchar(a);
  putchar(b);
  putchar(c);
  putchar(d);
}
```

其运行情况为：

```
good
```

说明：

- putchar（）函数只能用于单个字符的输出，并且一次只能输出一个字符；
- putchar（）函数在使用时，必须在程序（或文件）的开头加上编译预处理命令，即 #include "stdio.h"。

5. 单个字符输入函数

getchar（）函数的功能是从键盘输入一个字符，应用时在程序中必须用另一个变量接收读入的字符。

getchar（）函数的一般调用形式为：

```
c= getchar();
```

执行上面的调用时，变量 c 将得到用户从键盘上输入的一个字符值。这里可以是字符型或整型变量。

[例 9-5] getchar（）函数应用举例。

```
#include "stdio.h"
main()
{ char c;
  c=getchar();          /*接收用户从键盘上输入一个字符*/
  putchar(c);           /*输出字符型变量 c 的值*/
}
```

其运行情况为：

```
h
h
```

说明：

- getchar（）函数只能用于单个字符的输入，且一次只能输入一个字符；
- getchar（）函数在使用时，必须在程序（或文件）的开头加上编译预处理命令。

9.2.2　顺序结构程序设计举例

[例 9-6]　编程实现输入某三角形的底和高，计算其面积。

```
main()
{ float width,height,area;
  printf("Enter width and height:");
  scanf("%f,%f",&width,&heihgt);
  area=(width*height)/2.0;
  printf("The area is :%f\n",area);
}
```

其运行情况为：

```
Enter width and height :2.5,4
The area is :5.000000
```

[例 9-7]　某商场营业员工资的计算方法：每月 300 元的基本工资加该月总销售额的 8.5%的提成。要求编一程序，输入某营业员的月总销售额，计算并显示该营业员的月收入。

问题分析：设总销售额为 m，月收入为 salary，则 salary=300+m*8.5%。

其程序如下：

```
main()
{ float m,salary;
  printf("Enter total sales:");
  scanf("%f",&m);
  salary=300+m*0.085
  printf("salary is :%5.2f\n",salary);
}
```

其运行情况为：

```
Enter total sales:3500
Salary is :597.50
```

[例 9-8] 从键盘输入一个小写字母，输出其对应的大写字母及其相应的 ASCII 码值。

问题分析：从 ASCII 码表中得知，大、小写字母的 ASCII 码值相差 32，而 C 语言允许字符型数据和整型数据可以混合运算。因此，一个小写字母减去 32，即得到相应的大写字母。

其程序如下：

```
#include "stdio.h"
main()
{ char c1,c2;
  printf("Input a lower letter:");
  c1 = getchar();
  c2 = c1 - 32;
  printf("letter: %c,ASCII: %d \n",c1,c1);
  printf("letter: %c,ASCII: %d \n",c2,c2);
}
```

9.3 选择结构程序设计

在实际应用中，我们经常遇到类似这样的情况：如果某人的高考成绩高于（或等于）分数线，那么他就可以上大学；否则，他将落榜；在温控系统中，如果当前的温度高于正常温度范围，就启动降温装置；如果低于正常范围，就启动加热系统；如果在正常范围内，就无需进行任何操作；等等。这些问题的共同点是需要根据条件来做出判断，以得出某种结论，或根据条件的成立与否决定从两组或几组操作中选择一组执行。在程序设计中，用选择结构来实现条件判断的问题。

在 C 语言中，实现选择结构主要是 if 语句和 switch 语句。

9.3.1 if 语句

1. 不带 else 的 if 语句

语句格式：

```
if(表达式)语句
```

其执行过程：计算表达式的值；当表达式的值为真（非 0）时，执行语句；否则，什么也不做，顺序执行。

例如，if (a<b) a=b；条件表达式 a>b 的值为真，则执行 a=b；否则，即表达式值为假，不执行语句。这个条件语句执行完后，a 中存放的是 a，b 两数中较大的一个。

在程序设计中，当遇到某段语句只有在某种特定条件下才被执行时，就应该用 if 语句来控制。这段语句应变为复合语句并为 if 语句的内嵌语句。

[例 9-9] 根据输入的学生成绩对其进行判断处理：如果成绩及格（假定 60 分是及格线），则输出“通过”；否则，不做任何输出。

其程序如下：

```
main()
{float score;
 printf("Enter ascore:");
 scanf("%f",&score);
 if(score>=60.0)
    printf("通过\n");
}
```

其运行情况为：

```
Enter a score:50
(没有任何输出)
```

2. 带有 else 的 if 语句

语句格式：

```
if(表达式)语句 1 else 语句 2
```

其执行过程：计算表达式的值，表达式的值为真（非 0）时，执行语句 1，为假（0）时，执行语句 2。

该语句与不带 else 的语句是不同的。这里语句 1 和语句 2 必须根据条件选择其一执行。

[例 9-10] 输入 3 个整数，输出其中最大的一个。

要输出最大的一个，故必须将 3 个数进行比较。要比较，则先比较其中的两个，获得较大的，再去同第 3 个数比较，最后得到 3 个数中的最大者。要进行两个数的比较，则经常用到 if 语句，比较的结果可以作为下一步操作的条件。

其程序如下：

```
main()
{int a,b,c,max;
 printf("please input three numbers:");
 scanf("%d%d%d",&a,&b,&c);
 if(a>b)                              /*求 a,sb 中的大者存入变量 max 中*/
    max=a;
 else
    max=b;
 if(max<c)
    max=c;
 printf("The largerst is:%d",max);
}
```

上面程序也可以写成：

```
main()
{ int a,b,c,max;
   printf("please input three numbers:");
```

```
    scanf("%d%d%d",&a,&b,&c);
    if(a>b)
       max=a;
    else
       max=b;
    if(max>=c)
    printf("The largest is :%d",max);
    else
       printf("The largest is :%d",c);
}
```

[例 9-11] 输入 3 个数，按从小到大的顺序输出。

用变量 a，b，c 分别存放 3 个数，比较后，a 中存放最小者；c 中存放最大者；b 中存放中间者。

if（a>b）将 a 和 b 对调（a 是 a、b 中的小者）。

if（a>c）将 a 和 c 对调（a 是 a、c 中的小者，a 也就是 a、b、c 中的最小者）。

if（b>c）将 b 和 c 对调（b 是 b、c 中的小者，也就是三者中的次小者）。

其程序如下：

```
main()
{float a,b,c,t;                                    /*t 为两个变量值交换的中间变量*/
 printf("请输入 3 个实数(以逗号分隔)");
 scanf("%f,%f,%f",&a,&b,&c);
 if (a>b)
  {t=a;a=b;b=t;}                                   /*a,b 值互换*/
 if (a>c)
  {t=a;a=c;c=t;}                                   /*a,c 值互换*/
 if(b>c)
  {t=b;b=c;c=t;}                                   /*b,c 值互换*/
 printf("%10.2f,%10.2f,%10.2f",a,b,c);
}
```

在这个程序中，if 语句的内嵌语句为复合语句。复合语句中的 3 个赋值语句可变为一个逗号表达式，则程序为：

```
main()
{float a,b,c,t;
 printf("请输入 3 个实数(以逗号分隔)");
 scanf("%f,%f,%f",&a,&b,&c);
 if (a>b)
  t=a,a=b,b=t;                          /*逗号表达式构成的语句*/
 if (a>c)
  t=a,a=c,c=t;                          /*逗号表达式构成的语句*/
 if(b>c)
```

```
    t=b,b=c,c=t;                    /*逗号表达式构成的语句*/
    printf("%10.2f,%10.2f,%10.2f",a,b,c);
}
```

if 语句在使用时应注意：如果语句 1 或语句 2 不是复合语句，则其后一定要有一个分号。

3. if 语句的嵌套

所谓 if 语句的嵌套是指在 if 语句中又包含了一个或多个 if 语句。if 语句嵌套的目的是为了解决多路分支处理问题。在实际应用中，由于问题的复杂性，有时将 if 语句嵌在 if...else 结构中，有时将 if...else 结构嵌在 if 语句中，结果往往造成不知道哪一个 else 与哪一个 if 配对。因此，嵌套的 if 语句很容易出错，故建议采用下面的一种较为规范、容易掌握的嵌套的 if...else 结构形式。其流程图如图 9-1 所示。

```
if(表达式 1)
  语句 1
else if(表达式 2)
    else if (表达式 3)
          语句 3
          ……
        else if <表达式 n)
              语句 n
           else 语句 n+1
```

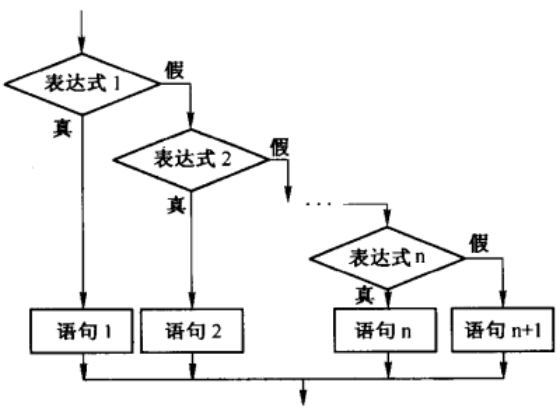


图 9-1　if 语句嵌套流程图

从流程图中可以看出，这种嵌套的 if 语句结构实现了多路选择处理：各个表达式依次求值，一旦某个表达式为真，那么就执行与之相关的语句，从而终止整个语句序列的执行。最后一个 else 部分用于处理上述条件都不成立的情况或默认情况。此时，上面的各个条件均不满足。

[例 9-12]　输入一个学生成绩，当成绩≥90 时，输出“Very good”；当 80≤成绩<90 时，输出“Good”；当 60≤成绩<80 时，输出“passed”；当成绩<60 时，输出“Failed”。

根据学生成绩分别落入不同的分数段（4 种）而输出不同的评语。这是一个具有 4 个分

支的问题。

其程序如下：

```
main()
{float score;
 printf("Enter a score: ");
 scanf("%f",&score);
 if(score>=90)
   printf("Very good! \n");
 else if (score>=80)
          printf("Good \n");
      else if (score>=60)
               printf("Passed \n");
           else printf("Failed");
}
```

其运行情况为：

9.3.2　switch 语句

if 语句通常用于实现两路分支选择，而在实际应用中，存在着大量的多路分支问题。尽管使用 if 语句的嵌套形式可以达到多路分支选择的目的，但随着问题复杂性的增加，分支越多，嵌套的层次数也就越多，这就必然会导致程序的可读性降低。因此，C 语言提供了 switch 语句来解决直接处理多路分支的选择问题。

switch 语句的一般形式为：

```
switch(表达式)
{ case  常量表达式 1:语句组 1;
  case  常量表达式 2:语句组 2;
  ……
  case  常量表达式 n:语句组 n;
  [default:语句组 n+1;]
}
```

switch 语句的执行过程：先计算表达式的值，然后顺次同 case 后的常量表达式的值比较，找到相等的常量表达式 i，则执行该常量表达式冒号后的语句组 i；若遇到 break 语句，则跳出 switch 结构，将程序流程转向 switch 结构的下一条语句执行；若没有 break 语句，则以该语句组为入口，依次执行其后的所有冒号后面的语句；如果 switch 语句后面的表达式的值找不到匹配的常量表达式的值，则此时要考虑是否有 default 可选项；若有，则执行 default后面的语句组直到结束；否则，什么也不执行。

[例 9-13]　用 switch 语句实现例 9-11。

```
main()
{ int score,mark;
  printf("Enter score");
  scanf("%d",&score);
  mark=-1;
  if(score>=0&&score<=100)
  mark=score/10;
  switch(mark)
  {case 10:
   case 9:printf("Very good\n");break;
   case 8:printf("Good \n");break;
   case 7:
   case 6:printf("passed");break;
   case 5:
   case 4:
   case 3:
   case 2:
   case 1:
   case 0:printf("Failed\n");break;
```

```
    default:printf("Enter error \ n");
}
```

switch 语句使用时要注意以下几点。

- switch 后面表达式的类型，一般为整型、字符型或枚举型；
- 每个 case 后面常量表达式的值必须互不相同，即不允许对表达式的同一个值有两种或两种以上的处理方案；
- 尽管各个 case 和 default 出现的次序不影响程序的执行结果，但是把 default 放在最后是一种良好的程序设计习惯；
- 多个 case 可共有一组执行语句，如例 9-13；
- case 后面的常量表达式仅起到语句标号的作用，并不进行条件判断，系统一旦找到入口标号，就从此标号开始执行下去，不再进行判断；
- case 语句又被称为开头语句，当仅需执行一个开头语句时，则在该 case 后面的语句中必须包含一个 break 语句，让其执行 break 语句来跳出 switch 语句，否则会一直执行到最后，或遇到下一个 case 中的 break 语句为止；
- 当 case 后面包含一条以上的执行语句时，可以不用花括号括起来，系统会自动识别并顺序执行所有语句；
- default 是可选项。但良好的程序设计习惯是在每个 switch 语句中，写上一条 default 语句来集中处理例外条件，从而可防止一些条件没有被测试到。

9.3.3 选择结构程序设计举例

[例 9-14] 有一函数 $y=\begin{cases}-1 & (x<0)\\ 0 & (x=0)\\ 1 & (x>0)\end{cases}$，编一程序，输入一个 x 值，输出 y 值。

程序一：用简单的 if 语句实现。

```
main()
{int x,y;
 printf("Enter the x:");
 scanf("%d",&x);
 if (x>0) y=1;
 if (x=0) y=0;
 if (x<0) y=-1;
 printf("x=%d,y=%d",x,y);
}
```

程序二：采用内层 if 嵌在外层的 if 后面的结构。

```
main()
{int x,y;
 printf("Enter the x"):
 scanf("%d",&x);
 if (x<=0)
```

```
    if (x==0)
        y=0;
    else
        y=1;
    else
        y=-1;
  printf("x=%d,y=%d",x,y);
}
```

程序三：采用内层 if 嵌在外层的 else 部分，即 else if 结构。

```
main()
{int x,y;
 printf("Enter the x:");
 scanf("%d",&x);
 if(x<0) y=-1;
 else if (x>0) y=1;
      else y=0;
 printf("x=%d,y=%d",x,y);
}
```

下面编写的程序是否正确？为什么？

```
main()
{int x,y;
 printf("Enter the x:");
 scanf("%d",&x);
 y=-1;
 if(x! =0)
   if (x>0) y=1;
       else y=0;
 printf("x=%d,y=%d",x,y);
}
```

[例 9-15]　某商店售货，按购买货物的款数多少分别给予不同的优惠折扣：

购货不足 250 元的，没有折扣；

购货满 250 元、不足 500 元的，折扣 5%；

购货满 500 元、不足 1000 元的，折扣 7.5%；

购货满 1000 元、不足 2000 元的，折扣 10%；

购货满 2000 元，折扣 15%。

分析：设购货款为 m，折扣为 d，根据题意则有：

$$d=\begin{cases}0 & (m<250)\\ 5\% & (250\leqslant m<500)\\ 7.5\% & (500\leqslant m<1000)\\ 10\% & (1000\leqslant m<2000)\\ 15\% & (20000\leqslant m)\end{cases}$$

方法一：用嵌套的 if 结构实现。

```
main()
{float m,d,amount;
 printf("Enter your money for buying: ");
 scanf(("%f",&m);
 if (m<250) d=0;
 else if (m<500) d=5;
      else if (m<1000) d=7.5;
           else if(m<2000) d=10;
                else d=15;
 amount=m*(1-d/100);
 printf("amount=%6.2f\n",amount);
}
```

方法二：用 switch 结构实现。

设置 mark 标记变量：mark=m/250，根据 mark 的值而确定相应的折扣。

其程序如下：

```
main()
{float m,d,amount;
 int mark;
 printf("Enter your money for buying:");
 scanf("%f",&m);
 mark=(int)(m/250);
 switch(mark)
 {case 0:d=0;break;
  case 1:d=5;break;
  case 2:
  case 3:d=7.5;break;
  case 4:
  case 5:
  case 6:d=10;break;
  default:d=15;
 }
 amount=m*(1-d/100);
 printf("amount=%6.2f\n",amount);
}
```

9.4 循环结构程序设计

计算机解决实际问题时经常遇到重复处理的内容，因此就需要在程序中将某些语句重复执行。一组被重复执行的语句被称为循环体。每重复一次都必须做出是继续重复还是停止重复的决定。这个决定所依据的条件被称为循环继续的条件。循环体与循环继续的条件一起构成了所谓的循环结构。C 语言提供了三种形式的循环控制语句，分别为 while 语句、for 语句和 do...while 语句。

9.4.1　while 语句

while 语句的一般形式为：

```
while(表达式)
语句
```

其中，表达式指明了循环执行的条件，一般是关系表达式或逻辑表达式，也可以是数值表达式。当表达式的值非 0 时，条件成立，执行语句（循环体）。循环体可以是单个语句，也可以是由用大括号括起来的多个语句构成的复合语句。

while 语句的执行过程是：先计算表达式的值；若值非 0，就执行循环体，当执行循环体之后，再计算表达式的值，决定是否再次执行循环体；若值为 0，则跳出结束循环语句，如图 9-2 所示。

[例 9-16]　打印 100 以内的所有奇数。

要打印 100 以内的所有奇数，首先设置变量 i 为 1，如果 i 小于 100，则输出 i，然后 i 每次增加 2，直到 i 大于 100 为止。

其程序如下：

```
main()
{int i=1;
 while(i<=100)
 {printf("%d",i);
   i+=2;
 }
 printf("\n");
}
```

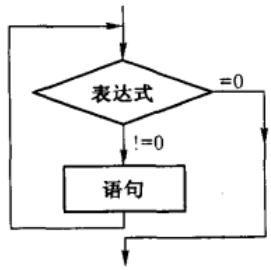


图 9-2　while 流程图

在执行 while 语句之前，变量 i 的初值为 1，进入 while 语句，首先计算表达式 $i\leqslant100$ 的值为 1，并判断为非 0，于是执行循环体：先打印 i 的值，然后 i 增加 2，使 i 变成 3，至此，一次循环结束；然后控制再转去计算表达式 $i\leqslant100$ 的值，由于这时 i 的值是 3，故 $i\leqslant100$ 的值为 1，再重复执行循环体；打印 i 的值 3，并使 i 的值变为 5，……如此重复，当 $i=99$ 时，表达式 $i\leqslant100$ 的值仍为 1，继续打印 i 的值 99，将使 i 增加 2，i 变为 101，本次循环结束，进入下次循环，计算表达式 $i\leqslant100$ 的值为 0，整个循环就宣告结束。

需要说明的是，while 语句执行结束后，程序中变量 i 的值是 101，而不是 99。

while 语句的特点是先判断表达式的值，然后才决定是否执行循环体。如果一开始表达式的值就是 0，那么循环体一次也不执行，如程序段：

```
i=10;
while(i<3)
printf("i=%d\n",i);
```

在这个 while 语句中，循环体一次也不执行。另一方面，当表达式的值恒为非 0 时，表示循环条件永远成立，循环就会无休无止地执行下去，则这种情况被称为“死循环”。在程

序设计中必须设法排除“死循环”。例如，

```
while(10)
{.....}
```

是一种“死循环”形式，但如果在循环体中有 break、return 或 goto 语句，则照样能正常结束循环。

[例 9-17] 求 $s=1^2+2^2+3^2+\cdots+n^2$ 前 n 项之和小于且最接近 10 000 时的项数 n 及前 n 项的和。

这是一个累加求和问题。用 s 表示级数前 i 项的和，k 表示级数前 $i+1$ 项的和，i 为级数项数计数。如果 $k\leqslant 10000$ 时，就进行累加；否则，结束循环。

其程序如下：

```
main()
{int s=0,k=1,i=0;
 while(k<=10000)
 {i++;
  s=s+i*i;
  k=k+(i+1)*(i+1);
 }
 printf("前%d项的和为s=%d\n",i,s);
}
```

在程序中，k 是前 $i+1$ 项之和。

其运行结果为：

```
前30项的和为9455。
```

9.4.2 for 语句

1. for 语句的一般形式

```
for(表达式1;表达式2;表达式3)
        语句
```

其中，表达式 1——循环初始化表达式，主要是对循环变量赋初值；表达式 2——控制是否继续循环的条件表达式，为真，则继续循环；否则，跳出循环；表达式 3——循环控制变量增量或减量的表达式；语句——循环体，循环过程中将重复执行。

其执行过程如下：

(1) 计算表达式 1 的值；

(2) 计算表达式 2 的值，若其值为真（非 0），则执行 (3)；若为假 (0)，则执行 (5)；

(3) 执行循环体；

(4) 计算表达式 3，返回 (2)；

(5) 跳出语句，执行其后继语句。

其执行流程图如图 9-3 所示。

[例 9-18] 在屏幕上输出所有可显示字符与其 ASCII 码的对照表。

分析：从 ASCII 码表中可以看出，所有可显示的字符是连在一起的，第一个字符是空格字符，ASCII 码为 32，最后一个字符为“～”，ASCII 码为 126。因此，只要输出 ASCII 码值的范围为 32～126之间的字符即可。

其程序如下：

```
main()
{char c;
 for(c=32;c<=126;c++)
 printf("字符%c 对应的 ASCII 码值为%d\n",c,c);
}
```

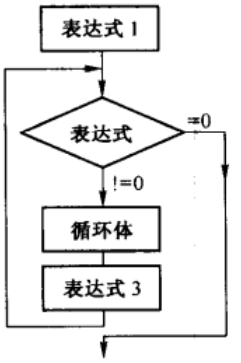


图 9-3　for 流程图

2．几点说明

for 语句中的表达式可以是任意表达式，如逗号表达式，可同时对多个变量赋初值及修改值，如 for（i=0，j=1；i<n&&j<n；i++，j++）。

for 语句中的 3 个表达式可以省略，但分号不能省略。

若省略表达式 1，如

```
i=1;
for(;i<4;i++)
  printf("%d\n",i);
```

与以下语句结果完全相同。

```
for(i=1;i<4;i++)
  printf("%d\n",i);
```

若省略表达式 3，如：

```
for (i=1;i<4;)
{ printf("%d\n",i);
  i++;
}
```

则这时循环体就不是单个语句，而是一个复合语句，必须用大括号括起来。

若省略表达式 2，则表达式循环条件永远成立，此时会出现死循环，除非在循环体中用其他手段（如 break，return 或 goto 语句等）来结束循环，如：

```
for(i=1;;i++)
  { printf("%d\n",i);
    if(i>d) break;
  }
```

若省略表达式 1 和表达式 3，则 while 语句与这样的 for 语句等价，即：

```
for(;表达式;)          while(表达式)
    语句                   语句
```

若3个表达式全省略，如：

```
for(;;)语句
```

则完全等价于while（1）语句。

注意：for（;;）；是一个“死循环”语句。

对for语句的功能完全可以用while语句实现。

```
for(表达式1;表达式2;表达式3)
        语句
```

等价于：

```
    表达式1;
while(表达式2)
 {语句
  表达式3;
 }
```

[例9-19] 输入整数 n，求 $n!$

这是个“连乘”的问题。为得到连乘的积，则程序中需要有：用来存放连乘积的变量（fac），其初值为1；控制连乘个数的循环控制变量（i），其初值为1，终值为 n，每次增量为1，每次要连乘的乘数，由于其恰好与循环控制变量（i）的值相吻合，因此可用循环控制变量的值代替。

其程序如下：

```
main()
{int i,fac,n;
 printf("please enter n:");
 scanf("%d",&n);
 fac=1;
 for(i=1;i<=n;i++)
 fac=fac*i;
 printf("%d! =%d\n",n,fac);
}
```

9.4.3 do...while语句

do...while语句的一般形式为：

```
do
  语句
while(表达式);
```

do...while语句的执行过程是：

（1）先执行语句；

(2) 计算表达式的值，根据其值判断，为真（非 0）时，返回（1）重新执行；

(3) 根据表达式判断，其值为假（即 0）时，结束 do...while 语句，执行其后继语句。

do...while 流程图如图 9-4 所示。

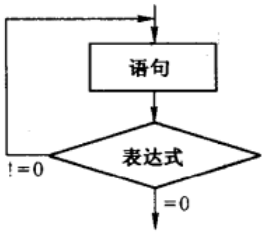


图 9-4　do...while 流程图

[例 9-20]　从键盘输入 1 个整数，把这个整数中的各位数字反序显示出来。例如，输入 2345，显示输出 5432。

要将一个整数反序显示出来，需先显示个位数，再显示十位数……其具体做法：该整数除以 10 的余数为个位数，再取该数除以 10 的商，得到一个新数，对这个新数进行与上述相同的操作，就可以求出各位数字。例如，2345，由 2345%10 得 5，2345/10 得 234，再由 234%10 得 4，234/10 得 23，再由 23%10 得 3，23/10 得 2，2%10 得 2，2/10 得 0。此时，所给整数的各位已逐一分离出来。循环终止的条件是某次求得商为 0 即可。

其程序如下：

```
main()
{int n,c;
 printf("please input a number:");
 scanf("%d",&n);
 do
 { c=n%10;
   printf("%d",c);
 }while(n/=10>0);
  printf("\n");
}
```

其运行情况为：

```
please input a number:2345
5432
```

9.4.4　几种循环的比较

三种循环都可以用来处理同一问题。在一般情况下，它们可以互相代替。

while 和 do...while 循环，只在 while 后面指定循环条件，在循环体中应当包含使循环趋于结束的语句（如 i++ 或 i=i+1 等）。

for 循环可以在表达式 3 中包含使循环趋于结束的操作，甚至可以将循环体中的操作全部放到表达式 3 中。因此，for 语句的功能更强，凡用 while 循环能完成的，用 for 循环都能实现。

用 while 和 do...while 循环时，循环变量初始化的操作应在 while 和 do...while 语句之前完成。而 for 语句可以在表达式 1 中实现循环变量的初始化。

while 循环、do...while 循环和 for 循环，可以用 break 语句跳出循环，用 continue 语句结束本次循环。

9.4.5 break 语句和 continue 语句

循环体使用 break 语句可以提前结束整个循环，退到循环体外；使用 continue 语句是为了提前结束本次循环而进入下一轮循环。

1. break 语句

break 语句的一般形式为：

```
break;
```

break 语句的作用：跳出循环语句或跳出 switch 语句。

break 只能用在循环语句和 switch 语句中，而且在循环语句中一般是与 if 语句一起配对使用的。

[例 9-21] 输入任意两个整数，求其最大公约数。

假设输入的两个整数分别是 8 和 12，最大公约数肯定位于 1～8 之间（包含 8），所以用 8 和 12 依次除 8，7，6，5，4，3，2，1，最先能够整除的数，则为最大公约数。显然当除到 4 时，后边的 3，2，1 就无须再除了，循环中止，最大公约数为 4。

其程序如下：

```
main()
{int a,b,min,i;
 printf("please intput integer a,b:");
 scanf("%d,%d",&a,&b);
 min=(a<b)? a:b;
 for(i=min;i>=1;i--)
   if (a%i==0&&b%i==0)
      break;
 printf("%d 和 %d 的最大公约数是%d",a,b,i);
}
```

2. continue 语句

continue 语句的一般形式为：

```
continue;
```

continue 语句的作用是结束本次循环，跳过循环体中尚未执行的语句，然后又去进行循环条件判断，决定是否进行下次循环。对于 for 语句，越过循环体中尚未执行的语句，执行计算表达式 3，计算表达式 2 并判断是否为真，以决定是否继续循环。

[例 9-22] 求 1～100 中所有能被 3 整除的数，并打印出来。

```
main()
{int i=1;
 while(i<100)
 {if (i%3!  =0)
    {i++;
```

```
            continue;
        }
        printf("%d",i);
        i++;
    }
    printf("\n");
}
```

在程序的循环体中，如果 i 不是 3 的倍数，则 i 增值，然后跳过后面语句，结束本次循环，提前进入下一轮循环；否则，打印 i 的值，继续下一轮循环。

9.4.6　循环嵌套

循环嵌套是指在一个循环体内包含了另一个完整的循环结构，又称为多重循环。3 种循环不仅可以自身嵌套，而且还可以互相嵌套。

下面是几种合法的循环嵌套形式：

```
(1) while ()
    {
     ......
     while ()
      {
        ......
       }
      ......
   }
(2) for ()
    {
     ......
     do
        {
          ......
        }
        while ()
      ......
   }
(3) while ()
    {
       ......
       for (;;)
        {
          ......
```

```
        }
    }
(4) do
      {
       ......
       for (;;)
        {
          ......
        }
      }
    while ()
```

[例 9-23] 打印乘法口诀表。

其程序如下：

```
main()
{int i,j;
  for(i=1;i<10;i++)
    {for(j=1;j<10;j++)
       printf("%10d * %2d= %2d",i,j,i*j);
    printf("\n");
    }
}
```

双重循环的执行过程是：先执行外循环，当外循环控制变量 i 取初值 1 后，执行内循环，在内循环 j 从 1 变化到 9 的过程中，i 始终不变，直到内循环执行完毕。到了外循环，i 才变为 2，而后再执行内循环，j 从 1 变化到 9，……如此循环下去，直到外循环控制变量 i 超过终值，整个双重循环才执行完毕。

9.4.7 循环结构程序设计举例

[例 9-24] 输入 10 个整数，求出其中的最大值和最小值。

设输入的数为 x，最大值为 max，最小值为 min。

首先输入第 1 个值并作为 max 与 min 的初值，然后每输入一个值，都将其与 max 和 min 进行比较。如果大于 max，则替代原来的 max 的值；如果小于 min，则替代原来的 min 的值。如此重复下去，max 始终存放的是当前已输入数的最大值，min 始终存放的是当前已输入数的最小值，直到第 10 个数输入完并比较完后，max 就是最大值，min 就是最小值。

其程序如下：

```
main()
{int i;
 int x,max,min;
 printf("Please input the first number:");
```

```
scanf("%d",&x);
max=x;
min=x;
for(i=2;i<=10;i++)
  {printf("please inout the next number");
   scanf("%d",&x);
   if (x>max)
      max=x;
   else if (x<min)
         min=x;
  }
printf("max=%d,min=%d",max,min);
}
```

[例 9-25]　利用格里高利公式求$\frac{\pi}{4}=1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\cdots$直到最后一项的绝对值小于等于$10^{-6}$为止。

总体来看，该题仍是一个“累加”问题，只不过每次要累加的数据一次是正数，一次是负数。因此，设置一个符号变量 sign 来控制每次要累加数据的符号。由于要重复累加的次数不能确定，因此不能采用计算器的方式来控制循环次数。但知道控制循环的条件是一个逻辑表达式，而且该表达式的值会在循环中被改变，因此采用 while 或 do...while 循环要比用 for 循环更加适宜。

其程序如下：

```
#include "math.h"
main()
{int sign=1;
 float t,pi;
 t=1.0;
 pi=0.0;
 do
 { pi=pi+t;
   n=n+2;
   sign=-sign;
   t=sign/n;
  }while(fabs(t)>=1e-6);
 pi=pi*4;
 printf("pi=%10.6f\n",pi);
}
```

其运行情况为：

```
pi=3.141594
```

[例 9-26]　输入一个整数 n（$n>3$），判断该数是否为素数。

数学方法已证明，若要判断一个整数 n 是否为素数，则只需用 $2\sim\sqrt{n}$ 各数去除 n。如果都除不尽，则 n 就是素数；只要有一次除尽了，则 n 就不是素数。

这里需要重复做的是用 $2\sim\sqrt{n}$ 去除 n，继续循环的条件是："除数未曾到$\sqrt{n}$并且未曾除尽过"。为此，需设一个开关变量 flag，来标识是否除尽。flag 初值为 1（真），一旦某一次除尽了就将其置为 0（假），从而终止循环。当循环结束时，若 flag 为真，就表明 n 是素数。

其程序如下：

```
#include "math.h"
main()
{int n,i,flag;
printf("Enter a integer:");
scanf("%d",&n);
i=2;
flag=1;
while(i<(int)sqrt(n)&&flag)
    if (n%i==0)flag=0;
    else i++;
if(flag) printf("%d is prime number.\n",n);
else printf("%d is not prime number.\n",n);
}
```

其运行情况为：

```
Enter a integer:78
78 is not prime number.
```

再运行：

```
Enter a integer:137
137 is prime number.
```

小结

本章包括顺序结构、选择结构和循环结构三部分内容。

顺序结构程序设计部分先简单介绍了 C 语言的语句，然后详细介绍了字符输入输出和格式输入输出函数，并通过具体的例题详细介绍了顺序程序的分析和实现，主要掌握常用的格式输入输出函数的应用，进而达到编写简单程序的目的。

选择结构程序设计部分主要介绍了 if 语句、swicth 语句及如何利用这两个语句实现分支程序。if 语句能描述任意的选择结构。switch 语句适合于特殊的多分支选择结构，即每个分支都能与某个常量相对应的选择结构。要重点掌握的是分析如何用选择结构描述带条件选择的问题，然后用关系表达式或逻辑表达式来表示条件，根据分析结果选用 if 语句或 switch 语句去实现它。

循环结构程序设计主要介绍了 while 语句、for 语句、do...while 语句和三种语句之间的相互嵌套，以及用三种语句解决有规律的重复操作问题。在通常情况下，while 语句适合

描述循环次数事先不确定但已知循环结束条件的问题；for 语句常用于编写循环次数确定的问题；do-while 语句更适合用于编写菜单选择程序。本节的关键在于学会分析具体问题，找出循环结构的组成要素，然后用循环语句实现。

习题

1. 设圆柱的半径 r 为 2.5，高 h 为 4，计算圆柱的表面积和体积。输出时要求有必要的文字说明，小数点后取两位数字。

2. 输入一个华氏温度，换算成摄氏温度并输出。换算公式为：$c=\frac{5}{9}(F-32)$。要求有输入提示信息和输出结果说明信息，取两位小数（提示：书写表达式时要考虑到计算时的类型转换，如 1/2 与 1.0/2 的区别）。

3. 输入直角三角形的两直角边的边长，计算三角形的周长和面积。

4. 写出下面程序的输出结果并上机运行，分析程序执行过程。

(1)
```
main ()
  { float c=3.0, d=4.0;
    if (c>d) c=5.0;
    else
    if (c==d) c=6.0;
    else c=7.0;
    printf ("\n %.1f", c);
  }
```

(2)
```
main ()
  { int t, a=6, b=10;
    if ( (a=b) >0) t=a+b;
    printf ("\n%d", t);
  }
```

(3)
```
#include "stdio.h"
main ()
  { int x=1, y=0, a=0, b=0;
    switch (x)
      {case 1:
       switch (y)
        { case 0: a++; break;
          case 1: b++; break;
        }
      case 2: a++; b++; break;
      case 3: a++; b++;
    }
    printf ("\na=%d, b=%d", a, b);
```

```
}
```

(4)
```
main ()
{ int n=0;
  while (n++<=2)
  printf ("%d\t", n);
  printf ("\n%d", n);
}
```

(5)
```
main ()
{ int p, n;
  for (n=0, p=1; p<=1000; n++, p*=n)
  printf ("\n n=%d, p=%d", n, p);
}
```

(6)
```
main ()
{ int i=1, s=3;
  do
  {s+=i++;
  if (s%7==0) continue;
  else ++i;
  } while (s<10);
  printf ("\n%d", i);
}
```

5. 有3个整数 a，b，c，由键盘输入，输出其中最小的数。

6. 有一个函数

$$y=\begin{cases} x & (x<1) \\ 2x-1 & (1\leqslant x<10) \\ 3x-11 & (x\geqslant 10) \end{cases}$$

写一程序，输入 x，输出 y 值。

7. 输入任意月份（1～12），输出该月份属于哪个季节。要求分别用 if 语句和 switch 语句实现。

8. 求一元二次方程 $ax^2+bx+c=0$ 的根。其中，a、b、c 由用户输入且 a 不等到于0。当 b^2-4ac 的值大于零、等于零、小于零时，分别计算方程的根。

9. 打印输出个位数为6并且能被3整除的所有的5位数。要求用 while、do...while、for 语句分别实现。

10. 输入一行字符（以回车符结束），分别统计其中字母、数字、空格及其他字符的个数。

11. 计算 e 的近似值：$e=\frac{1}{1!}+\frac{1}{2!}+\frac{1}{3!}+\cdots+\frac{1}{n!}$（设 $n=10$）

提示：编程时要注意 1/2 与 1.0/2 的区别。

12. 编程求满足公式关系的最大 n：$12+22+32+\cdots+n2\leqslant 10000$

提示：该题的特点是循环次数不确定，但已知循环结束的条件。

13. 取红、黑、白球；盒子里共有 12 个球，其中红球有 3 个，黑球有 3 个，白球有 6 个，若从中任取 8 个球，则至少包含一个红球的取法有多少种？并且输出每一种取法。

14. 鸡兔同笼问题：笼子里有鸡、兔若干只，其总头数为 16，脚的总数为 40，问笼子里有鸡、兔各有多少只？

15. 打印输出斐波那契数列的前 20 项。该数列有如下特点：前两个数为 1、1，从第三个数开始，则是其前面两个数之和，即

$$F_1 = F_2 = 1 \qquad (n \leqslant 2)$$

$$F_n = F_{n-1} + F_{n-2} \qquad (n \geqslant 3)$$

16. 打印出所有的“水仙花数”。所谓“水仙花数”是指一个 3 位数，其各位数字的立方和等于该数本身。例如，153 是一水仙花数，因为 $153 = 1^3 + 5^3 + 3^3$。

17. 一个数如果恰好等于它的因子之和，则这个数就是“完数”。例如，6 的因子为 1、2、3，则 6＝1＋2＋3，因此 6 是“完数”。编程序找出 1000 之内的所有完数，并按下面格式输出其因子：

6　its factors　are　1，2，3

第 10 章　数　　组

本章要点

- ☑ 一维数组的定义、初始化和引用
- ☑ 选择法排序
- ☑ 二维数组的定义、初始化和引用
- ☑ 字符数组的定义、初始化和引用
- ☑ 字符串函数

数组是相同数据类型元素的集合，用统一的数组名来表示，每一个元素用下标来区分。数组元素称为下标变量，而以前的变量则称为简单变量。在实际应用中，当需要处理大量同类型的数据时，利用数组是很方便的。

与使用变量一样，C 语言规定，程序中用到的数组必须先进行定义。数组说明包括要说明数组的名字、类型、维数与大小。

10.1　一维数组

10.1.1　一维数组的定义

定义一维数组的一般形式如下：

```
类型说明符　　数组名[常量表达式];
```

例如，定义语句：

```
int a[10],b1[45];
double xy[20];
```

共定义了 3 个一维数组，整型一维数组 a，共包括 10 个元素（a [0] ～a [9]），数组中的每一个元素均为整型；整型一维数组 b1，共包括 45 个元素（b1 [0] ～b1 [44]），其中的每一个元素也都是整型；双精度实型一维数组 xy，共包括 20 个元素（xy [0] ～xy [19]），其中的每一个元素均为双精度实型。

关于数组的定义要注意以下几个问题：

- 数组名的命名规则与变量名相同；
- 常量表达式必须为整型，并且用方括号括起来（不能用圆括号）；
- 常量表达式中可以包含符号常量，但不能是变量。

例如，下面的数组定义是允许的：

```
#define N 20
main()
{int c[N+5];
……
}
```

其中，N 是符号常量，代表整数 20。因此，定义的整型数组 c 共有 25 个元素（c [0] ~ c [24]）。

下面的数组定义是不允许的：

```
main()
{int n;
scanf("%d",&n);
int a[n];
……
}
```

数组类型可以是任何基本类型，即整型、实型、字符型，也可以是将在后面学到的指针类型，构造类型等。

必须注意，在 C 语言中，数组元素的下标是从 0 开始的，如

```
int a[15];
```

定义了一个长度为 15 的整型一维数组，在这个数组中的 15 个元素分别为 a [0]，a [1]，… a [14]，其中并不包含元素 a [15]。

10.1.2　一维数组的引用

和其他类型的变量一样，数组也必须先定义，再引用。一维数组元素的引用形式为

```
数据名[下标]
```

设有定义：float f [8]，则 f [0]，f [1]，f [5+1]，f [i] 都是合法的引用方法（这里 i 是整型变量），而 f [-3]，f [8]，f [5.6] 是非法的引用方式。

说明：

- 一个数组元素就是一个变量，它的使用规则与同类型的普通变量相同。
- 不要企图通过引用数组名来引用整个数组，只能引用单个的数组元素。在 C 语言中，数组名有特殊的含义。数组名代表数组的起始地址。这一点将在本章后面几节的内容中详细介绍。
- 定义数组时，数组名后面的方括号中的内容和引用数组元素时数组名后面的方括号中的内容的含义是不同的，前者是数组的长度，而后者是元素的下标。
- 数组元素的下标可以是常量、变量、表达式，但下标的值的类型必须是整型的，而且必须是在合法的范围内，长度为 N 的数组的下标的范围是 0，1，2，…，$N-1$。
- C 程序运行时，编译系统并不检查数组元素的下标是否越界。若下标越界，则可能会产生严重的后果，所以需要编程人员在编程时自己保证数组的下标不要越界。

10.1.3 一维数组的初始化

可以在定义数组时完成数组的初始化，如

```
int arr[10]={0,1,2,3,4,5,6,7,8,9};
```

说明：

- 将数组元素的初始值放在一对花括号中，所赋初值的类型必须与定义数组的类型一致，各个初值之间用逗号分隔，系统将按初值的赋值顺序，顺次给数组元素赋值。
- 所赋初值的个数不能超过数组元素的个数（数组的长度），但可以少于数组元素的个数，这时相当于给数组的前一部分元素赋值，后面的元素由系统自动地赋值为 0，如：

```
int b[6]={1,2,3};那么 b[0]=1,b[1]=2,b[2]=3,b[3]=0,b[4]=0,b[5]=0
```

- 若给数组的所有元素赋初值，则可以省略数组的长度，如：

```
int w[]={3,4,5,7,9,10};
```

系统会根据所赋初值的个数确定数组的长度。该例中数组 w 的长度为 6。

- 若是在数组定义之后再初始化数组，则不能成组赋值，只能一次一个元素赋值，例如，下面语句是错误的赋值方式。

```
int w[8];
w[8]={1,3,5,7,4,6,7,9};
```

10.1.4 一维数组应用举例

[例 10-1] 从键盘输入 10 个整数，然后分别以正序和逆序输出。

要给数组赋值和输出数组，则要从头到尾逐个访问数组中的每一个元素，使用一个循环语句，使循环控制变量从小到大变化就可以实现；如果要逆序输出数组，则要从数组尾开始，逐个访问数组元素即可。

其程序如下：

```
main()
{int a[10];
 int i;
 printf("Please input 10 numbers:");
 for(i=0;i<10;i++)
 scanf("%d",&a[i]);
 printf("\n");
 for(i=0;i<10;i++)
  printf("%d",a[i]);
 printf("\n");
 for(i=9;i>=0;i--)
  printf("%d",a[i]);
}
```

[例 10-2]　用选择法对 10 个数排序（从小到大）。

选择法排序的思路是：每次从待排序的数据中选择最小的数，与相应位置上的数交换。假设待排序的数据存放在一维数组 a 中，那么第 1 次从 10 个元素中选择最小数与 a [0] 交换，第 2 次从剩下的 9 个元素中选择最小的数与 a [1] 交换……直至从剩下的两个数中选择最小的与 a [8] 交换。这样通过选择 9 次，达到排序的目的。

例如，有待排序的 10 个整数 3，5，1，10，4，8，23，12，0，-3。

第 1 趟排序（i=0）的过程是：

```
3, 5, 1, 10, 4, 8, 23, 12, 0, -3      初始化 min=0
↑
min
3, 5, 1, 10, 4, 8, 23, 12, 0, -3      j=1 时，a [j] >a [min]，min 不变
↑
min
3, 5, 1, 10, 4, 8, 23, 12, 0, -3      j=2 时，a [j] <a [min]，min 指向 j
   ↑
  min
3, 5, 1, 10, 4, 8, 23, 12, 0, -3      j=3 时，a [j] >a [min]，min 不变 j
   ↑
  min
3, 5, 1, 10, 4, 8, 23, 12, 0, -3      j=4 时，a [j] >a [min]，min 不变 j
   ↑
  min
3, 5, 1, 10, 4, 8, 23, 12, 0, -3      j=5 时，a [j] >a [min]，min 不变 j
   ↑
  min
3, 5, 1, 10, 4, 8, 23, 12, 0, -3      j=6 时，a [j] >a [min]，min 不变 j
   ↑
  min
3, 5, 1, 10, 4, 8, 23, 12, 0, -3      j=7 时，a [j] >a [min]，min 不变 j
   ↑
  min
3, 5, 1, 10, 4, 8, 23, 12, 0, -3      j=8 时，a [j] <a [min]，修改 min 指向 j
                        ↑
                       min
3, 5, 1, 10, 4, 8, 23, 12, 0, -3      j=9 时，a [j] <a [min]，修改 min 指向 j
                           ↑
                          min
-3, 5, 1, 10, 4, 8, 23, 12, 0, 3
```

在第 2 趟排序（i=1）的过程中，同样可通过比较得到最小元素的下标放在 min 中，即

3，5，1，10，4，8，23，12，0，3

↑

min

这时，min=8，说明第 i 个元素不是最小元素，需要交换，结果为

-3，0，1，10，4，8，23，12，5，3

在第 3 趟排序（$i=2$）过程中，同样可通过比较得到最小元素的下标放在 min 中，即

-3，0，1，10，4，8，23，12，5，3

↑

min

这时，min=i，说明第 i 个元素已在合适的位置，不需交换。

……

最后的结果为

-3，0，1，3，4，5，8，10，12，23

其程序如下：

```
#define N 10
#include "stdio.h"
main()
{int a[N],i,j,min,temp;
 printf("\nPlease enter %d number:\n",N);
 for(i=0;i<N;i++) scanf("%d",&a[i]);
 for(i=0;i<N;i++)
{min=i;
for(j=i+1;j<N;j++)
    if(a[j]<a[min]) min=j;
if(min!=i)
   {temp=a[i];a[i]=a[min];a[min]=temp;}
   }
printf("\nThe sorted array is :\n");
for(i=0;i<N;i++)
    printf("%6d",a[i]);
}
```

10.2 二维数组

10.2.1 二维数组的定义

二维数组定义的格式为：

```
类型说明符    数组名[常量表达式 1][常量表达式 2],……
```

例如：

```
float f[3][4];
```

表示定义了一个第一维长度为 3、第二维长度为 4，数组元素为实型的二维数组。这个二维数组共有 12 个（3×4）个数据元素。

二维数组中，数组名的命名、数据类型及数组长度的选取要求与一维数组相同。

数组名后面紧跟的两个方括号里面分别存放数组的第一维长度和第二维长度，数组元素的个数就是两维长度之积。

可以认为二维数组是一维数组的延伸：二维数组是特殊的一维数组，它的元素类型又是一个一维数组。如上例中的数组 f 可以看作是含有 3 个元素 f [0]，f [1]，f [2] 的一维数组，而每一个元素又是一个含有 4 个实型元素的一维数组。

二维数组名代表二维数组的起始地址，在上面的定义中，f 即代表数组 f 的起始地址，而 f [0]，f [1]，f [2] 并不是真正的数组元素，而是分别代表第 0 行、第 1 行、第 2 行的起始地址。

从逻辑上看，二维数组元素间的关系相当于矩阵；第一维长度是矩阵的行数，第二维长度是矩阵的列数。从存储上看，内存是一维的、线性的空间，那么就要采取一定的方式来存储二维数组，即先存放第 0 行的元素，再存放第 1 行的元素，……同一行按列的顺序存放。

二维数组存放的规律为：第二维下标变化快，第一维的下标变化慢。

10.2.2　二维数组元素的引用

二维数组的引用方式为：

```
数组名[下标 1][下标 2];
```

设有定义："int b [6] [3]"；那么，b [0][2]、b [3][4]、b [3+2][0]、b [i][j]（i、j 为整型变量）都是合法的引用方式；而 b [2][3]、b [1.2][2]、b [6][0] 是非法的引用方式。

说明：

- 二维数组的下标范围和一维数组类似，可以是整型的常量、变量、表达式。第一维下标的范围是 0，1，2，…，第一维长度 - 1；第二维下标的范围是 0，1，2，…，第二维长度 - 1。
- 虽然二维数组是元素为一维数组的特殊的一维数组，但并不能引用数组的行或列，只能引用单个数组元素。
- 二维数组元素的表示方式不能写为如b[1,2]，b[2]，[0]，b(1,2) 的形式，而且两个下标之间也不能有空格。

10.2.3　二维数组的初始化

二维数组初始化的方式有如下几种。

给所有的元素赋初值，每一行的初值放在一个花括号中，所有行的初值再放在一个花括号中，如：

```
int  a[2][3]={{1,2,3},{4,5,6}};
```

上面的语句是将第 1 个花括号内的数据依次赋值给 a [0][0]，a [0][1]，a [0][2]，将第 2 个花括号内的数据依次赋值给 a [1][0]，a [1][1]，a [1][2]。

给部分元素赋初值，则剩余的元素自动被赋值为 0，如：

```
int  c[4][4]={{1,2,3,4},{4,5,6},{7,8}};
```

表示对数组的第 0，1，2 行的部分元素赋初值。同时，元素 c [1][3]，c [2][2]，c [2][3] 及第 3 行所有元素被自动赋值为 0，可以对中间某些行不赋值，如

```
int  c[4][4]={{1,2,3,4},{},{},{9}};
```

表示对数组的第 0，3 行元素赋初值，第 1，2 行元素被自动赋初值为 0。

若赋初值时，省略表示行的花括号，如：

```
int  a[3][3]={1,2,3,4,5};
```

则系统自动按数组 a 的元素在内存的顺序，将初值赋给相应的元素

```
a[0][0]=1,a[0][1]=2,a[0][2]=3,a[1][1]=4,a[1][2]=5;
```

其余元素被自动赋值为 0。

在赋初值时，如果指定了所有的行，那么定义数组时可以省略第一维的长度。编译系统会根据赋初值的情况，自动得到第一维的长度，如：

```
int  a[][3]={{1,23,3},{4}};
```

编译系统能够分析出第一维的长度是 2。

若是在定义二维数组后，再给其元素赋值，则需要逐个赋值，不能成组赋值。

10.2.4 二维数组应用举例

[例 10-3] 矩阵的转置：输入一个 3 行、4 列的矩阵，转置后输出。

所谓的矩阵的转置是指将矩阵的行、列互换所形成的新的矩阵。

其程序如下：

```
main()
{ int a[3][4],b[4][3];
  int i,j;
  printf("\nPlease input array a(3*4):\n");
  for(i=0;i<3;i++)
    for(j=0;j<4;j++)
      scanf("%d",&a[i][j]);
  for(i=0;i<3;i++)
    for(j=0;j<4;j++)
      b[j][i]=a[i][j];
  printf("\nThe array b(3*4):");
  for(i=0;i<4;i++)
    {printf("\n");
```

```
        for(j=0;j<3;j++)
            printf("%3d",b[i][j]);
        }
}
```

其运行结果为：

```
Please input array a:
0 1 2 3 4 5 6 7 8 9 10 11
The array b:
    0 4 8
    1 5 9
    2 6 10
    3 7 11
```

[例 10-4]　在矩阵中，若有元素在它所在的行中值最小，而在它所在的列中值最大，则称这样的元素为鞍点。编程序查找一个矩阵中的鞍点并输出。

先在矩阵中按行比较，找出这一行中最小的值，然后在这个元素的列上进行比较，如该元素最大，则这个元素是鞍点，输出该元素。

其程序如下：

```
#define M 5
#define N 6
main()
{ int a[M][N];
  int i,j,k,m;
  int flag=0;
  printf("\nPlease input array :\n");
  for (i=0;i<M;i++)
    for(j=0;j<N;j++)
        scanf("%d",&a[i][j]);                    /*输入数组元素*/
  for (i=0;i<M;i++)
    { m=0;
    for(j=1;j<N;j++)
      if(a[i][j]>a[i][m]) m=j;                   /*找当前行的最小元素,m 指向其列下标*/
    for(k=0;k<M;k++)
        if (a[k][m]>a[i][m])                     /*判断该元素在其所在的列是否最大*/
            break;
    if(k>=M)
      {flag=1;printf(("\n%d,%d,%d",i,m,a[i][m]);}  /*若该元素是鞍点,输出*/
    }
    if(flag==0)printf("\nnone.");
    }
```

10.3 字符数组与字符串

将用于存放字符型数据的数组称为字符数组。字符数组中的一个元素存放一个字符。

10.3.1 字符数组的定义

定义字符数组的一般形式为：

```
char 数组名[常量表达式]                    一维字符数组
char 数组名[常量表达式 1][常量表达式 2];    二维字符数组
```

例如：

```
char c[14];
c[0]='G';c[1]='o';c[2]='o';c[3]='d';c[4]=' ';c[5]='b';c[6]='y';c[7]='e';c[8]='!';
```

字符数组经赋值后，在计算机内存中的存放形式如图 10-1 所示。

c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]	c[8]
G	o	o	d		b	y	e	!

图 10-1 字符数组在内存中的存放形式

10.3.2 字符数组的初始化

对于字符数组的初始化，最容易理解的方式是逐个字符赋值给数组中的各元素，如：

```
char c[9]={'G','o','o','d',' ','b','y','e','!'};
```

把 9 个字符分别赋给 c[0]～c[8] 9 个元素。

如果花括号中提供的初值个数（即字符个数）大于数组长度，则按语法错误处理。如果初值个数小于数组长度，则只将这些字符赋给数组中前面的那些元素，其余的元素自动定为空字符（即‘\0’），如：

```
char c[11]={'G','o','o','d',' ','b','y','e','!'};
```

其数组 C 状态图如图 10-2 所示。

c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]	c[8]	c[9]	c[10]
G	o	o	d		b	y	e	!	\0	\0

图 10-2 字符数组 c 状态图

如果提供的初值个数与预定的数组长度相同，在定义时可以省略数组长度，则系统会自动根据初值个数确定数组长度。

[例 10-5] 从键盘输入一行字符，存放在字符数组中，然后逆序输出。

其程序如下：

```
main()
{char a[80];
 int k=0,j;
 printf("\nPlease input the chars:");
 scanf("%c",&c);
 while(c! ='\n')
 {a[k++]=c;
 scanf("%c",&c)}
 printf("\n");
 for(j=k-1;j>=0;j--)
    printf("%c",a[j]);
}
```

10.3.3　字符串和字符数组

1. 字符串的标志

字符串是作为一个整体对待的字符序列。在 C 语言中，没有字符串这种数据类型，而是将字符串存放在字符数组中。

在用字符数组存放字符串时，一般不关心数组的具体长度，而是关心有效字符的长度，因此一般预先定义数组要足够长。对于字符串，并不显式保存它的长度，而是用字符结束标志‘\0’来表示字符串到此结束，隐含着字符串的长度。

不但用字符数组存放字符串采用‘\0’表示字符串的结束，对于字符串常量，也采用字符串结束标志‘\0’来作为结束符。编译系统在处理字符串时会自动在字符串的最后一个字符之后加上‘\0’。也就是说，对于含有 n 个字符的字符串在内存中占 $n+1$ 个字节的空间，因为需要用一个字节来存放字符串结束标志‘\0’。

注意： 在处理字符串常量的时候，字符串结束标志是编译系统自动添加的，不用人为地在字符串最后加上‘\0’。

2. 将字符串常量赋给字符数组

可以将字符串常量中的字符逐个赋值，但最后一个字符是空字符，如：

```
char c[20]={'T','h','i','s','','a','','s','t','r','i','n','g','\0'};
```

可以在定义字符数组时同时将字符串常量赋值给它，如：

```
char c[20]={"This a string"};
```

习惯上可以省略花括号，则上面的定义可写为：

```
char c[20]="This a string";
```

这两种定义方式和初始字符数组的方式是等效的。

有时候，定义字符数组可以省略数组的长度，系统会根据所赋字符串常量的实际长度来确定字符数组的长度。例如，上例定义可以写为：

```
char c[ ] = "This a string";
```

系统会自动确定数组 c 的长度为 14。

若在字符数组定义后再初始化，则不能整体赋值，只能给每个元素逐个赋值。

10.3.4 字符串处理函数

C 语言提供了一些专门处理字符串的库函数，但在调用这些库函数时，需要包含头文件“string.h”。

1. 字符串输出函数 puts（字符数组）

字符串输出函数 puts 的作用是将一个字符串输出到终端。该函数没有返回值，如：

```
char str[ ] = "This is a string"
puts(str)
```

其输出为：

```
This is a string
```

2. 字符串输入函数 gets（字符数组）

字符串输入函数 gets（字符数组）的作用是接收从键盘输入的一个字符串，存放在字符数组中。函数的返回值是字符数组的起始地址。

设已定义：“char str [80]”；则 gets（str）；表示程序运行时接收从键盘输入的字符串。

若输入：“string”那么，字符数组 str 中就接收了这个字符串。它的存储图如图 10-3 所示。

s	t	r	i	n	g	\0	\0	\0	\0
0	1	2	3	4	5	6	7	8	9

图 10-3 字符数组 str 的存储图

3. 字符串比较函数 strcmp（字符串 1，字符串 2）

字符串比较函数 strcmp（字符串 1，字符串 2）的作用是比较两个字符串的大小。这里的字符 1、字符串 2 可以是字符串常量，或者是已赋值的字符数组。

它的返回值是整型的。若字符串 1 大于字符串 2，则返回一个正数；若字符串 1 等于字符串 2，则返回 0；若字符串 1 小于字符串 2，则返回一个负数。

字符串的比较方法是：从左到右依次比较两个字符串中的对应字符，若相等，则继续比较，直到当前两字符不相等，或遇到‘\0’结束。若全部字符都相等，则认为两字符串相等，返回 0；若不相等，则以第一对不等字符为准得到比较结果。字符比较实际上是比较它们的 ASCII 码的大小。

4. 字符串复制函数 strcpy（字符数组 1，字符串 2）

字符串复制函数 strcpy（字符数组 1，字符串 2）的作用是：将字符串 2 的内容原样复制到字符数组 1 中去。函数的返回值是字符数组 1 的起始地址。字符串 2 可以是字符串常量或者是已经赋值的字符数组；字符数组 1 是已定义的足够长的字符数组。

C 语言不能直接将字符串常量或字符数组赋值给一个字符数组，只能通过字符串复制函数 strcpy 来实现。

5. 字符串连接函数 strcat（字符数组 1，字符数组 2）

字符串连接函数 strcat（字符数组 1，字符数组 2）的作用是：将字符串 2 连接到字符数组 1 中的字符串后，将连接后的新字符串存放于字符数组 1 中。函数返回值是字符数组 1 的起始地址。

字符串 2 可以是字符串常量，或者是已赋值的字符数组；字符数组 1 是已定义的足够长的字符数组。连接时，要去掉字符数组 1 中字符串的结束标志‘\0’，再把字符串 2 连接到它的后面，新字符串最后保留一个‘\0’。

6. 测试字符串长度函数 strlen（字符数组）

测试字符串长度函数 strlen（字符数组）的功能是求字符数组中存放的字符串的长度，返回值就是求得的长度。函数的参数要求是字符串常量或已赋值的字符数组。

字符串的长度指的是字符串的实际长度，而不包含字符串结束标志‘\0’在内。

[例 10-6] 输入一个由字母组成的字符串，再分别以大写字母和小写字母的形式输出该字符串。

```
#include "stdio.h"
main()
{ char c[60];
  int k;
  printf("\n Please input the string:");
  gets(c );
  printf("\n");
  k=0;
  while (c[k]! ='\0')
    {if(c[k]>='a' && c[k]<='z') putchar(c[k]-32);
     else putchar(c[k]);
     k++ }
  printf("\n");
  k=0;
  while(c[k]! ='\0')
    {if(c[k]>='A' &&c[k]<='Z') putchar(c[k]+32);
     else putchar(c[k]);
     k++ ;}
}
```

小结

数组是有序数据的集合。数组的特点是用一个统一的数组名和变化下标可以访问数组中的任何一个元素。当一组数据的数据类型相同时，使用数组操作处理这组数据比较方便。

本章包括一维数组、二维数组及字符数组的应用三部分内容，主要介绍了三种数组的定义、初始化和引用方法。其中需要重点掌握的是数组元素的引用及利用数组所构成的算法。

习题

1. 用冒泡法对 10 个整数排序。

2. 用数组求裴波那契数列中前 20 个数。

3. 求一个 3×3 的整型矩阵对角线元素之和。

4. 已有一个已经排好序的数组，今输入一个数，要求按原来排序的规律将其插入数组中。

5. 打印出以下的杨辉三角形（要求打印出 10 行）。

```
1
1  1
1  2  1
1  3  3  1
1  4  6  4  1
⋮  ⋮  ⋮  ⋮  ⋮
```

6. 打印“魔方矩阵”。所谓魔方矩阵是指这样的方阵，它的每一行、每一列和对角线之和均相等。例如，三阶魔方矩阵为：

```
8  1  6
3  5  7
4  9  2
```

7. 有 n 个学生，每人考 m 门课，要求：

(1) 找出成绩最高的学生号和课程号；

(2) 找出有不及格课程的学生号及其各门课程的全部成绩；

(3) 求各门课程的平均分数。

8. 求矩阵 a（3 行 2 列）和 b（2 行 2 列）的乘积，结果存入矩阵 c 中，并按矩阵形式输出。

9. 找出方阵每行、每列中最小元素及其所在的行号。

10. 有 15 个数，由大到小的顺序存放在一个数组中，输入一个数，要求用折半查找法找出该数是数组中第几个元素的值。如果该数不在数组中，则打印出“无此数”。

11. 编写一个程序，将两个字符串连接起来，不要用 strcat 函数。

12. 编写一个程序，将字符数组 s2 中的全部字符复制到字符数组 s1 中。不用 strcpy 函数。复制时，‘\0’也要复制过去。‘\0’后面的字符不复制。

第 11 章　函　　数

本章要点

- ☑ 函数的定义和调用
- ☑ 函数的嵌套调用和递归调用
- ☑ 函数调用的数据传递方式
- ☑ 局部变量和全局变量
- ☑ 内部函数和外部函数
- ☑ 编译预处理

结构化程序设计的要点是自顶向下、逐步细化及模块化。在程序设计过程中，一个大的程序按功能可分成若干模块，每个模块又可由更小的模块所构成，这样逐层细化至只完成一个特定的功能小模块。采用这样的原则编写的程序，层次清晰，可读性强，可重用性好，易于修改和维护，亦便于多个人员合作编写。

C 语言由函数来实现模块。函数是完成特定功能的一段程序，是 C 程序的基本单位。一个 C 程序由一个或多个源程序文件构成，每个源程序文件可以由一个或多个函数构成，而且每个 C 源程序文件可为多个 C 程序共享。较大的程序可放在多个源程序文件中，以便于分别编写、编译，提高效率。

11.1　函数的定义

11.1.1　函数的概述

一个 C 程序至少由一个 main 函数（又称主函数）构成，一般由一个 main 函数和若干其他函数构成。这些函数可以放在一个文件中，也可以放在多个文件中。在编译时，用 #include语句将各文件包含进来。所有函数的定义是并列的、独立的，函数定义之间没有嵌套或从属关系。函数之间可以互相调用。

在 C 语言中，函数可分为以下两种。

(1) 标准库函数

这种函数用户不必定义，可直接使用。例如，scanf ()，printf ()，gets ()，puts () 等都是 C 语言中常用的库函数。

(2) 用户自己定义的函数

这种函数用以解决用户的专门问题，一般由用户自己编写。

[例 11-1]　从键盘输入两个正整数 m，n（$m \geqslant n$），求 $\frac{m!}{(m-n)!}$ 的值（即求 P_m^n）。

其程序如下：

```
#include "stdio.h"
main()
{  int m,n;
    int  p();
    scanf("%d,%d",&m,&n);
    if(m>=n)  printf("%d\n",p(m)/p(m-n));
    else printf("m<n!! \n");
}
int  p(k)
  int k;
{  int s,i;
   s=1;
   for(i=1;i<=k;i++)        s=s*i;
   return(s);
}
```

这个程序共有两个函数：一个主函数 main（），它的功能只是从键盘上输入两个正整数 m 和 n，然后当 m≥n 时，通过调用函数 p（）计算并输出 p（m）/p（m－n）的值；另一个是函数 p（），它就是用户自定义函数，其功能是计算阶乘值，如 p（m）是计算 m!，p（m－n)是计算（m－n)!。

11.1.2 函数的定义

在 C 语言中，函数定义的一般形式为

```
类型标识符号 函数名(形参列表)
形参类型说明;
{  说明部分
   语句部分
}
```

在定义 C 函数时要注意以下几点。

① 函数名符合标识符的命名规则，用来唯一标识一个函数，同一程序中的函数不能重名。函数名一般用有意义的名字表示，以提高程序的可读性。

② 函数类型标识符同变量类型说明符，表示返回的函数值类型。此外，C 语言还可以定义无类型（即 void 类型）的函数。这种函数不返回函数值，而只是完成某种功能。

③ 如果省略函数的类型标识符，则默认为 int 类型。例如，在例 11-1 中，可以省略前面的类型标识符 int，也就是函数定义为：

```
p(k)
  int k;
{  int s,i;
   s=1;
```

```
    for(i=1;i<=k;i++)          s=s*i;
    return(s);
}
```

④ C 语言允许定义空函数，如：

```
dummy()  {  }
```

⑤ 如果“形参列表”中有多个形式参数，则它们之间要用逗号“,”分隔。

⑥ C 语言允许在形参列表中直接对形参的类型进行说明。例如，在例 11-1 中定义的函数 p（）中，可以定义为：

```
p(int  k)
{  int s,i;
   s=1;
   for(i=1;i<=k;i++)      s=s*i;
   return(s);
}
```

11.1.3　函数参数和返回值

1. 函数的参数

当被调用函数有参数时，则主调函数和被调函数间有数据传递关系。

定义函数时的参数被称为形式参数，简称形参。形参在该函数未被调用时是没有确定的值的参数，只是形式上的参数。调用函数时的参数被称为实参。实参可以是变量、常量或表达式，是有确定值的参数，是实实在在的参数。函数定义时，形参不占内存，只有发生被调用时，形参才能被分配内存单元，接受实参传来的值。

定义函数时必须定义形参的类型。函数的形参与实参要求在个数上相等，并且对应的形参和实参的类型相同。形参和实参可以同名。形参是局限于该函数的变量，即使形参和实参同名，也是两个不同的变量，占用不同的内存单元。

2. 函数的返回值

函数是完成特定功能的程序段。主调函数通过函数调用完成一定的功能。有时调用函数的目的是为了得到一个计算结果。这就是函数的返回值。

函数的返回值是通过被调用函数中的 return 语句得到的。其格式为：

```
return（表达式）;
```

它的执行过程是先计算表达式的值，再将计算结果返回给主调函数。

函数返回值的类型应为定义函数时函数值的类型。若函数值的类型与 return 语句中表达式的值类型不一致，则以函数值类型为准。

为提高程序的正确性，没有返回值的函数应用 void 定义其函数值类型；否则，函数将带回一个不确定的值。

return 语句的另一项功能是结束被调用函数的运行，返回到主调函数中继续执行后继语句。在无返回值的函数中也可能有 return 语句。其格式为：

```
return;
```

这里，return 语句用以结束函数的执行，返回主调函数。函数中无 return 语句，那么函数会执行到函数体后的“}”处才结束。

11.2 函数的调用

11.2.1 函数的调用

函数调用即主调函数通过传递一定的信息来使用被调用函数的功能。

函数调用的一般形式是：

```
函数名(实际参数表列)
```

若被调函数是无参函数，则实参表列为空。

调用函数的过程是：为函数的所有形参分配内存单元，计算各个实参表达式的值，一一对应地赋值给相应形参（若是无参函数，则上述过程不执行）；进入函数体，执行函数中的语句，实现函数的功能，执行到 return 语句时，计算 return 语句中表达式的值（若无返回值函数，则本项不做），释放形参及本函数内局部变量所占的内存，返回主调函数；继续执行主调函数中的后继语句。

[例 11-2] 计算两个实数的和与积，并将和与积按大小顺序输出。

```
main()
{void  output();
float add(float c,float  y);
float  multiply(float x,float y);
void sort(float c,float y);
float a,b;
float sum,mul;
output();
printf("\nPlease enter  data(a,b):");
scanf("%f,%f",&a,&b);
sum=add(a,b);
mul=multiply(a,b);
sort(sum,mul);
output();
}
void  output()
  {printf("\n**********************************");}
float add(float x,float y)
  {return(x+y);}
float mutiply (float p,float q){return(p*q);}
void sort(float  c,float  y)
  {float  a,b;
```

```
    if(x>=y){a=x;b=y;}
    else {a=y;b=x;}
    printf("\nthe max is :%f,the min  is %f:",a,b);
return;
}
```

本例中包含 5 个函数 main，output，add，multiply，sort。其中，main 函数调用其他 4 个函数。这里，函数 output，add，multiply，sort 出现在主调函数 main 之后，且返回值非整型，所以必须在 main 函数中加以声明。这个程序执行时先调用函数 output。output 是一个无参无返回值的函数。它的功能是输出一行星号，然后输入两个实数分别存放于变量 a，b 中，以 a、b 为实参调用函数 add 计算 a 与 b 的和，返回值赋给 sum，再以 a 与 b 为实参调用函数 multiply 计算 a 与 b 的积，返回值赋给 mul；接下来以 sum 与 mul 为实参调用函数 sort，比较 sum 与 mul 的大小并按大小顺序输出；函数 sort 无返回值，执行完 return 语句后返回 main 函数。

函数调用的方式有如下 3 种。

- 函数调用可单独成为一个语句，如例 11-1 中的“output ();”，这里只要求函数完成一定的功能，不需要返回值。
- 调用函数后的返回值可参加表达式的计算，如例 11-2 中的“sum = add (a, b);”和“mul = multiply (a, b);”，这时要求被调函数带返回值。
- 带返回值的函数调用亦可作为其他函数的参数。

11.2.2　函数的声明

在一函数中调用另一函数需要具备以下条件。

- 被调用的函数必须是已经存在的函数，即被调用函数或者是库函数或者是用户自定义的函数。
- 对于库函数，要在主调函数所在文件中用 # include 命令将相应的头文件包含进来。
- 如果使用用户自定义函数，则要在主调函数中对其进行声明。

进行函数声明的目的是为了使编译系统在编译阶段对函数的调用进行合法性检查，判断形参与实参的类型及个数是否一致。若没有声明函数就直接调用，那么只有在运行时才会发现实参与形参不匹配的错误，就需要重新调试程序，加大工作量。

对特定的函数，定义函数是对函数功能的确立，只有一次，而声明是对已定义函数的返回值进行类型说明，可以有多次，主调函数在声明部分对被调函数进行声明，即通知编译系统，下面将要调用某函数。

对函数声明采用函数原型的方法。函数原型的格式为：

```
类型标识符　函数名(形式参数表列)
```

若无参数，则形参表列为空，但“()”不能省略。简单地说，函数原型即是照写定义的函数首部，再加上分号，如例 11-2 中 output (), float add (float x, float y); float multiply (float x, float y); 等等。

在实际应用中，以下 3 种情况可以省略对被调函数的声明：

- 若被调函数的定义出现在主调函数之前，则编译系统已预先知道了函数的类型，可

根据函数首部对函数的调用进行检查；

- 被调用函数的返回值类型是整型或字符型的，因为整型是系统默认的类型；
- 在所有函数定义之前，在函数的外部已做了对函数的声明，则主调函数内可以不必再次声明。

除上述 3 种情况外，主调函数在调用函数前必须对被调函数进行声明。

11.3 函数的嵌套调用和递归调用

11.3.1 函数的嵌套调用

在 C 语言中，函数的定义是独立的，函数与函数之间无从属关系，不允许嵌套定义，但函数之间可以嵌套调用，即在调用一个函数的过程中，可以又调用另一个函数。

[例 11-3] 按下列公式计算 e 的近似值，n 的值由键盘输入。

$$e=1+\frac{1}{1!}+\frac{1}{2!}+\frac{1}{3!}+\cdots+\frac{1}{n!}$$

分析：这是一个累加求和的问题，只是这个累加比较特殊，每个加数的分母是一个数的阶乘，而该数在前一个数基础上自动增加 1。先做一个函数求一个数的阶乘，再做一个函数，计算累加求和，在该函数中调用求阶乘函数。

其程序如下：

```
main()
{double sum(int m);
 long fac(int p);
 int n;
 float s;
 printf("\nPlease input n:");
 scanf("%d",&n);
 s=sum(n);
 printf("\ne=%f",s);
}
double  sum(int  m)
{  double  s1=1;
int j;
for(j=1;j<=m;j++)
    s1+=1.0/fac(j);
return(s1);
}
long  fac(int p)
{long  f=1;
int j;
for(j=1;j<=p;j++)
    f*=j;
```

```
return(f);
}
```

本函数的运行过程如图 11-1 所示。

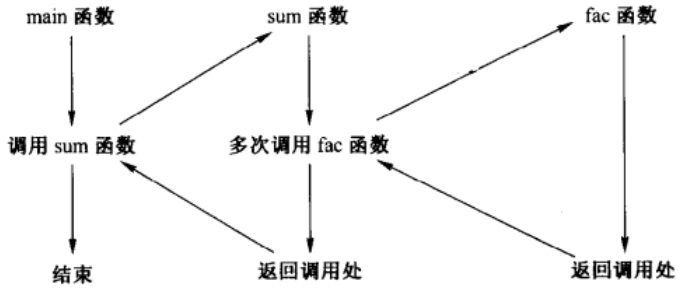


图 11-1　例 11-3 函数的运行过程

这个程序的执行过程是：执行 main（）函数的开头部分；执行到“s＝sum（n);”时，转去执行 sum（）函数；将 n 的值赋值给形参 m，执行 sum（）函数的开头部分；j 从 1 到 m，反复执行：

- 在“s1＋＝1.0/fac（j);”中以 j 为实参调用函数 fac（），转去执行函数 fac（）；
- 执行函数 fac（），fac（）函数中未调用其他函数，执行至结束；
- 返回到函数 sum（）中调用 fac（）函数处，计算 s1 的值。

执行 sum（）函数的后继语句，即返回 main（）函数；执行 main（）函数的剩余语句，直到结束。

11.3.2　函数的递归调用

在调用一个函数的过程中又出现直接或间接地调用该函数本身的过程被称为函数的递归调用。图 11-2 是函数递归调用的示意图。

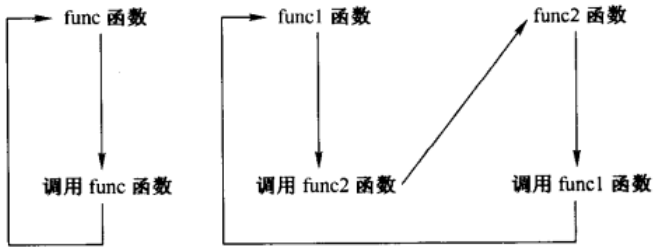


图 11-2　函数递归调用的示意图

从图中可以看到，这两种递归调用都是无终止的自身调用。显然，程序中不应出现这种无终止的递归调用，而应是有限次数的、有终止的递归调用。这可以用 if 语句来控制，只有在某一条件成立时才继续执行递归调用；否则，就不再继续。

[例 11-4]　用递归求 $n!$

```
main()
{long  fact(int n);
```

```
int n;
printf("\nPlease enter n:");
scanf("%d",&n);
printf("\n%d! = %ld",n,fa<t(n));
}
long fact(int n)
{ if(n<=1) return(1);
  else return(n * fact(n-1));
}
```

[例 11-5] 汉诺塔问题。有 A、B、C 3 个底座，上面可以放盘子。初始 A 座上有 n 个盘子，这些盘子大小不相同，大盘子在下，小盘子在上，依次排列。要求将 A 座上 n 个盘子移至 C 座上，每次只能移动一个，并要求移动过程中保持小盘子在上，大盘子在下，可借助 B 座移动。编程输出移动步骤。

这个问题可用递归的思想来分析，将 n 个盘子由 A 座移动到 C 座可分为如下 3 个过程：

- 先将 A 座上 $n-1$ 个盘子借助 C 座移至 B 座；
- 再将 A 座上最下面的一个盘子移至 C 座；
- 最后将 B 座上 $n-1$ 个盘子借助 A 座移至 C 座。

上述过程是把移动 n 个盘子的问题转化为移动 $n-1$ 个盘子的问题，按这种思路，再将移动 $n-1$ 个盘子的问题转化为移动 $n-2$ 个盘子的问题……，直至移动一个盘子。

可以用两个函数来描述上述移动过程：

- 从一个底座上移动 n 个盘子到另一底座；
- 从一个底座上移动 1 个盘子到另一底座。

其程序如下：

```
void move_1(char from,char to)
  {printf("\n%c->%c",from,to);
  }
  void move_n(int m,char one,char two,char three)
  {if(m==1)move_1(one,tree);
     else {move_n(m-1,one,three,two);
           move_1(one,three);
           move_n(m-1,two,one,three);
          }
}
main()
{ int n;
  printf("\nPlease enter the number of disk:");
  scanf("%d",&n);
  printf("\nThe step of moving %d disks :",n);
  move_n(n,'A','B','C');
}
```

11.4　函数调用的数据传递方式

在 C 语言中，函数调用是值传递方式，即函数的实际参数和形式参数之间的数据传递方向是单向的，只能由实际参数传递给形式参数，而不能由形式参数传递给实际参数。在内存中，形式参数与实际参数占用的是不同的内存单元。当调用函数时，给形式参数分配内存单元，将实际参数的值赋值给形式参数。调用后，形式参数单元释放，实际参数仍保留调用前的值，形式参数值的变化不影响实际参数。

一般来说，形式参数与实际参数的结合方式有以下两种。

(1) 数值结合（单向传递）

这种方式在被调用函数中改变了形参值，并不改变调用函数的实参值。这种结合方式中的实参可以是变量，也可以是表达式。

在 C 语言中，当形参为简单变量时，均采用数据值结合。在这种情况下，一个函数只能通过函名返回一个函数值。前面所举的例子都是用这种方式来实现参数传递的。

[例 11-6]　交换两个参数的值。

其程序如下：

```
main()
{ void swap(int x,int y);
  int a,b;
  printf("\nPlease input data(a,b):");
  scanf("%d,%d",&a,&b);
  swap(a,b);
  printf("\na=%d,b=%d",a,b);
}
void swap(int x,int y)
{ int t;
  t=x;x=y;y=t;
  printf("\nx=%d,y=%d",x,y);
  return;
}
```

其运行结果为：

```
Please input data(a,b):4,8
x=8,y=4
a=4,b=8
```

(2) 地址结合（双向传递）

在这种方式中，如果被调用函数中改变了形式参数值，则同时也就改变了调用函数中的实参值。但这种结合方式中的实参只能为变量。

以地址作为传递参数的方式有用数组名作为参数和指针作为参数两种。

前面学习数组时，数组名是数组起始地址，因此用数组名做参数，实际上就是地址结合。

[例 11-7]　用选择法对数组中 10 个整数按由小到大排序。

其程序如下：

```
void  sort(int array[],int n)
{ int i,j,k;
  for(i=0;i<n-1;i++)
     {k=i;
       for (j=i+1;j<n;j++)
         if(array[j]<array[k])  k=j;
         if(k!  =i)
          {t=array[i];array[i]=array[k];array[k]=t;}
      }
    return;
     }
main()
{int a[10],i;
 printf("\nenter the  array\n");
 for(i=0;i<10,i++)
   scanf("%d",&a[i]);
sort(a,10);
for(i=0;i<10;i++)
   printf("%d",a[i]);
    printf("\n");
}
```

经过运行后，可以看到在执行函数调用语句“sort（a，10);”之前和之后，a 数组中各元素的值是不同的。原来是无序的，执行后，a 数组已经排序了。这是由于形参数组 array 已经采用选择法进行排序了，形参数组改变了，则实参数组也随之改变了。

11.5 局部变量与全局变量

C 语言明确规定：变量在使用前必须先定义。每个变量通常包含三方面的属性，即存储类型、作用域和生存期。

11.5.1 变量的作用域

变量的作用域就是变量的有效范围。变量的作用域由变量定义语句出现在程序中的位置决定，并据此划分为全局变量和局部变量。

全局变量：在所有函数之外定义的变量被称为全局变量。全局变量的作用域是从定义变量处开始到程序结束这一段区域。有一种说法，称全局变量为外部变量。全局变量的使用增加了函数之间传递数据的途径，在全局变量的作用域内的任何函数都能引用该全局变量，一个函数对全局变量的修改，能影响到其他引用这个变量的函数，因此全局变量的使用不当会产生意外的错误。

全局变量的使用，也会使得函数的通用性降低。从结构化程序设计方法的角度看，函数

应是完成单一功能的程序段，过多使用全局变量，会使函数之间的依赖性增加、耦合性高。在通常情况下，除非函数有特别要求，建议避免使用全局变量。

局部变量：在函数体内或复合语句内定义的变量称为局部变量。局部变量的作用域是定义该变量的函数体内或复合语句内。在它的作用域外，局部变量是不可见的，也就是说，一个函数内定义的局部变量是不能被其他的函数所引用的。

局部变量的特性有助于实现信息隐藏，即使不同的函数定义了同名的局部变量，也不会相互影响。

[例 11-8]　全局变量与局部变量的使用。

其程序如下：

```
int  a=1, b=2;
int  func1 (int x)
{x+ =a++ ;
b++ ;
return (x);
}
int c=3;
int  func2 (int x, int y)
{int a;
a=x+y;
b++ ; c++ ;
return (a);
}
main ()
{int u=4, v=5;
int s1, s2;
printf ("\n a=%d, b=%d, c=%d, u=%d, v=%d", a, b, c, u, v);
H: {char c='A';
int  u='a';
     c + =32;
     printf ("\nc=%c, u=%d", c, u);
}
printf ("\nc=%d, u=%d", c, u);
s1=func1 (u);
printf ("\na=%d, b=%d, c=%d, s1=%d", a, b, c, s1);
s2=func2 (u, v);
printf ("\na=%d, b=%d, c=%d, s1=%d", a, b, c, s2);
}
```

其运行结果为：

```
a=1,b=2,c=3,u=4,v=5
c=A,u=97
```

```
c=3,u=4
a=2,b=3,c=3.s1=5
a=2,b=4,c=4,s2=9
```

注意：

- 不要企图在主调函数中使用被调用函数的内部变量；
- 为了程序的可读性好，建议尽少用同名变量；
- 为了与局部变量有明显的区别，一般约定全局变量的第一个字母大写。

11.5.2 变量存储类别

C程序运行时占用的内存空间通常分为3部分，如图11-3所示。程序运行期间，数据分别存放在静态存储区和动态存储区。静态存储区用来存放程序运行期间需占用固定存储单元的变量，如外部变量和静态类别的内部变量。动态存储区用来存放不需要长期占用内存单元，可根据需要动态地分配和回收空间的内部变量，如内部变量、函数形参与函数调用时的返回地址等。

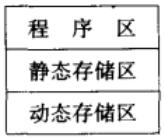


图11-3 内存空间分配

变量的存储类别分为两类，即动态存储类别和静态存储类别。

动态存储类别的变量：该类变量当进入定义它的函数或复合语句时被分配存储空间，当离开时所占内存空间被释放。动态存储类别是一种节省内存的存储方式，在需要时被建立，不需要时被撤销。动态存储类别有两种存储方式：auto（自动）类型的变量被存放在内存的动态存储区，register（寄存器）类型的变量被存放于寄存器中。

静态存储类别的变量：在源程序编译的时候被分配空间，从程序开始执行，一直占用固定的内存空间，直到程序运行结束，才会释放所占内存空间。静态存储类别的变量存放于内存的静态存储区中，可以用static（静态）、extern（外部）说明符来定义和声明变量。

11.5.3 局部变量的存储类别

局部变量的作用域是定义它的函数或复合语句。局部变量的存储类型是指它存放的位置。局部变量可存放于内存的动态区、寄存器和内存的静态区。但无论局部变量存放在何处，它的作用域是不变的。

1. auot（自动）变量

auto变量的存储单元被分配在内存的动态存储区中。在函数内部，auto存储类型是系统默认的类型，所以在某函数内部，下面两种定义的效果是相同的：

```
int  a;
auto  int  a:
```

当进入函数体或复合语句时，系统自动为自动变量分配存储单元，退出时自动释放这些存储单元另做它用。

2. register（寄存器）变量

register变量的存储单元被分配在寄存器中。register变量的作用域和生存周期与自动变量相同。只有局部自动变量和形式参数可以定义为寄存器变量。通常把使用频率较高的变量

定义为 register 类型。

一台计算机中寄存器的数量是有限的，而且寄存器的数据长度也是有限的，所以定义 register 变量的个数不能太多，数据类型太大的变量（如数组、结构体等数据类型的变量）也不适合定义为 register 变量。

register 的声明通常是不必要的。当今优化的编译系统能够识别频繁使用的变量，并能够在不需要编程人员做出 register 声明的情况下就把这些变量放到寄存器之中。

3. static（静态）变量

static 变量的存储单元被分配在内存的静态存储区中。static 变量在编译的时候被分配内存、赋初值，并且只被赋初值一次，对不赋值的内部 static 变量，系统自动给它赋值为 0。在整个程序运行期间，static 变量在内存的静态存储区占用着固定的内存单元，即使它所在的函数被调用结束后也不释放存储单元。它所在单元的值也会继续保留。因此下次再调用该函数时，static 变量仍使用原来的存储单元中的值。可以利用 static 变量的这个特点，编写需要在被调用结束后仍保存内部变量的值的函数。

用 static 声明的内部变量的作用域仍然是定义该变量的函数体或复合语句内部。虽然 static 变量在整个程序运行期间都是存在的，但在它的作用域外是不可见的。也就是说，其他函数是不能引用它们的。

[例 11-9]　编程序测试函数内部 auto 变量和 static 变量在每次函数调用时的内容。

其程序如下：

```
#include  "stdio.h"
void test_n_s()
{auto   int auto_v=0;
 static int static_v=0;
 printf("\nauto_v=%d, static_v=%d",auto_v, static_v);
 auto_v++;
 static_v++;
}
main()
{   int i;
    for(i=0;i<3;i++)
    test_n_s();
}
```

函数的运行结果为：

```
auto_v=0; static_v=0
auto_v=0; static_v=1
auto_v=0; static_v=2
```

11.5.4　外部变量的存储类型

1. 外部变量的定义

外部变量能存放在内存的静态存储区。外部变量在整个程序的运行期间占用内存单元，

而且一直起作用。外部变量分为两种定义方式，即程序级外部变量的定义和文件级外部变量的定义。

(1) 程序级外部变量的定义方式

程序级外部变量是在函数体外定义的变量，定义时不加任何存储类型的声明。所谓程序级的外部变量是指它的作用域可以是整个程序，只需加上特定的声明，就可以被所在程序的其他文件所使用。

(2) 文件级外部变量的定义方式

文件级外部变量也是在函体外定义的变量，定义时需用 static 声明。文件级外部变量的作用域是它所在的文件。虽然它在程序的运行期间一直存在，但它们不能被其他文件使用，用 static 声明的外部变量能够限制它的作用域的扩展，达到信息隐蔽的目的。

2. 外部变量的声明

外部变量的声明是扩展和限制外部变量的作用域。

(1) 用 extern 声明外部变量

在同一个文件中，定义在后、使用在前的外部变量，在使用前应用 extern 对其进行声明。

在多个程序文件中，使用其他文件中定义的外部变量，在本程序中应用 extern 对其进行声明。

[例 11-10] 假设一个程序包括两个文件 p1 _ 30.c 和 p2 _ 30.c。文件 p2 _ 30.c 中使用 p1 _ 30.c 中的外部变量，因此要在文件 p2 _ 30.c 中对该外部变量进行声明。

其程序如下：

```
/* p1 _ 30.c */
#include "p2 _ 30.c"
extern int Max,Min;
main()
{void func(int x,int y);
 int a,b,c;
 printf("\nplease input data(a,b,c):");
 scanf("%d,%d,%d",&a,&b,&c);
 func(a,b,c);
 printf("\n Max=%d,Min=%d",Max,Min);
}
/* p2 _ 30.c */
int Max;
void func(int x,int y,int z)
{extern int Min;
 Max=x;
 Min=x;
 if (y>Max) Max=y;
 if(z>Max) Max=z;
 if(y<Min) Min=y;
```

```
    if(z<Min) Min=z;
}
int Min;
```

在文件 p2 _ 30.c 中使用 Min 在定义它之前，需要在函数中先声明才能使用。在文件 p1 _ 30.c中要使用 p2 _ 30.c 中定义的外部变量 Max、Min，需要在使用前声明。

(2) 用 static 声明外部变量

在一个 C 程序中，如果有的外部变量只允许本文件中使用而不允许其他文件使用，则可以用 static 进行声明。此时，这个变量作为静态外部变量。

11.6　内部函数和外部函数

函数本质上是全局的，因为一个函数要被另外的函数调用。但是，也可以指定函数不能被其他文件调用。根据函数能否被其他源文件调用，可将函数分为内部函数和外部函数。

11.6.1　内部函数

如果一个函数只能被本文件中的其他函数所调用，则称其为内部函数。在定义函数时，在函数名和函数类型前加 static。其格式为

```
static　类型标识符　函数名(形参表)
```

例如：

```
static int fun(int x,int y)
```

内部函数又被称为静态函数。使用内部函数可以使函数只局限于所在文件，如果在不同的文件中有同名的内部函数，则将互不干扰。这样，不同的人可以分别编写不同的函数，而不用担心所用函数是否会与其他人编写的文件中函数同名。通常把只能在同一文件使用的函数和变量放在一个文件中，在它们前面都冠以 static 使之局部化，其他文件不能引用。

11.6.2　外部函数

外部函数是可以被程序中的其他文件所调用的函数。其定义格式为：

```
extern 数据类型 函数名(形式参数表列)
{声明部分;
 执行部分;
}
```

在调用时，先在文件的头部用 extern 声明所用的函数是外部函数，然后才能正常调用。

外部函数是 C 语言的默认函数类型。若没有特别的声明为 extern 类型，则系统也会默认为外部函数。

[例 11-11]　外部函数的使用。该程序由两个文件 p1.c 和 p2.c 组成。

其程序如下：

```
/* p1.c */
```

```
#include "p2.c"
#include "stdio.h"
main()
{extern char next(char c);
 char ch;
 while((ch=getchar())! ='\n')
   printf("\n%c,%c",ch,next(ch));
}

/* p2.c */
char next(char c)
{char d;
 d=c+1;
 return(d);
}
```

11.7 编译预处理

“编译预处理”，顾名思义，就是编译之前预先处理。也就是在对C语言源程序进行编译之前，由编译处理程序先对编译预处理命令进行处理。

C语言规定，所有的编译预处理命令均以“#”开头，每条预处理命令的结尾不能加分号，以便区别于C语句，作用范围是从定义处到本程序结尾。

C语言提供了三种编译预处理命令。

11.7.1 宏替换

宏替换写在源程序的前面，分为两种，即不带参数宏替换和带参数宏替换。

1. 不带参数宏替换

命令格式：

```
#defin 宏名 宏体
```

其中，宏名是用户定义的标识符，通常为大写，以区别于C语言中的变量。

宏体是任意的字符串，作用：从定义处开始，凡是出现宏名的地方均用宏体替换。

例如：

```
define N 15
```

编译时，预处理命令把源程序中所有出现N的地方都用15替换，系统再对替换后的源程序进行编译。

2. 带参数的宏替换

命令格式：#define 宏名（形参表） 宏体

其中，宏名是用户定义的标识符；宏体是含有形参表中的形参的表达式，替换时由语句的实参替换。

例如：

```
#define  M(x,y)  x * y
.....
c=M(2,3);                /* 使用带参宏名 */
```

替换后：

```
c= 2 * 3;
```

3. 宏定义的终止

命令格式：

```
#undef  宏名
```

作用：提前终止指定宏名的作用范围。

11.7.2　文件包含

文件包含是指把另一个文件的内容包含于源文件中。

文件包含是模块化程序设计的一种手段，可以把一些常用的宏定义、数据及函数等集中起来，单独构成一个文件，用包含命令把它们包含在所需的文件中。

命令格式：

```
#include  "文件名"
```

或

```
#include  <文件名>
```

作用：用指定文件的内容替换该命令行。

如果用尖括号把文件名括起来，则系统到存放库函数头文件所在的目录中录找要包含的文件，被称为标准方式；若用双引号把文件名括起来，则系统首先在源文件所在的目录下查找指定的文件，如果找不到，则再按照标准方式到有关目录中去查找。

在前面的例子中，已经使用到包含命令。

[例 11-12]　文件包含应用例子。

```
#include  "math.h"
main()
 {float  x,y;
    x=4.0;
    y=sqrt(x);
   printf("y= %f \ n",y);
 }
```

如果程序中没有 #include "math.h"，就会出现错误。

11.7.3　条件编译

条件编译是指让编译程序根据条件有选择地对源程序的一部分进行编译。

条件编译的三种命令形式。

```
(1) #if 常量表达式 1
      程序段 1
    #elif  常量表达式 2
      程序段 2
    #else  程序段 3
    #endif
```

当常量表达式 1 为真时，编译程序段 1；否则，常量表达式 2 为真时，编译程序段 2；否则，编译程序段 3。其中，#elif 可以省略，也可以用#elif 继续嵌套。

```
(2) #ifdef  标识符
      程序段 1
    #else
      程序段 2
    #endif
```

当标识符被定义过时，编译程序段 1；否则，系统编译程序段 2。

```
(3) #ifndef  标识符
      程序段 1
    #else
      程序段 2
    #endif
```

当标识符未被定义过时，编译程序段 1；否则，编译程序段 2。

小结

函数是 C 语言的基本组成单位，利用函数可以实现各模块的功能，可以减少重复编写程序的工作量。

本章主要包括函数的应用、局部变量和全局变量及预编译处理三部分内容。

在函数应用中，主要介绍的是函数的基本概念、如何定义函数、如何调用函数、函数的嵌套及函数递归调用，其中应重点掌握的是如何定义函数和调用已定义的函数。所有函数在定义时是互相独立的，不能嵌套定义，但可以嵌套调用，函数调用时要注意形参与实参类型和个数的匹配，在函数递归调用时，应有递推公式和递归结束条件。

在局部变量和全局变量部分，主要介绍的是局部变量与全局变量的基本概念、作用域及不同存储类型变量的作用域。编程时应注意变量的作用域。

在预处理中，主要介绍的是 3 种常用预处理命令，即宏定义、文件包含和条件编译及各自的作用，应重点掌握在编写程序时，如何有效地利用预处理命令，改进程序设计环境，提高编程效率。

习题

1．写两个函数，分别求两个整数的最大公约数和最小公倍数，用主函数调用这两个函

数，并输出结果，两个整数由键盘输入。

2．编写一判断素数的函数，在主函数输入一个整数，输出是否为素数的信息。

3．编写一个函数，使给定的一个二维整型数组（3×3）转置，即行、列互换。

4．编写函数 int mymod（int a，int b），求 a 被 b 除后的余数。

5．编写函数，求 $1-3+5-7+\cdots+n$。提示：函数应该有一个参数，通过这个参数得到 n 的值。

6．编一个函数，重复打印给定的字符 n 次。

7．写出计算 Ackermann 函数 Ack（m，n）的递归计算函数。对于 $m\geqslant 0$，$n\geqslant 0$，Ack（m，n）定义为

$$\begin{aligned}&\mathrm{Ack}(0,n)=n+1\\&\mathrm{Ack}(m,0)=\mathrm{Ack}(m-1,1)\\&\mathrm{Ack}(m,n)=\mathrm{Ack}(m-1,\mathrm{Ack}(m,n-1))\end{aligned}$$

8．编写一个递归函数，求出斐波那契数列的第 n 项。

9．编写一个函数，把任意十进制正整数转换成二进制数。要求：在主函数中进行输出，而且不能按逆序输出。提示：把十进制数不断被 2 除，余数放在一个一维数组中，直到商数为 0。

10．编写函数，求 $1-1/2+1/3-1/4+1/5-1/6+\cdots+1/n$。

11．分别编写出计算 Hermite 多项式 Hn（x）值的递推和递归函数。Hn（x）定义为

$$\begin{aligned}&\mathrm{H0}(x)=1 \qquad (x=0)\\&\mathrm{H1}(x)=2x \qquad (x=1)\\&\mathrm{Hn}(x)=2x\mathrm{Hn}-1(x)-2(n-1)\mathrm{Hn}-2(x) \qquad (\mathrm{x}>1)\end{aligned}$$

13．编写一个函数，输入一行字符，将此字符串中最长的单词输出。

14．定义一带参数的宏，使两个参数的值互换，并写出程序，输入两数作为使用宏时的实参，输出已交换后的两个值。

15．分别用函数和带参的宏，从 3 个数中找出最大数。

第 12 章　指　　针

本章要点

- ☑ 指针和指针变量的基本概念
- ☑ 指针变量的定义和初始化
- ☑ 指针的基本运算：&、*
- ☑ 指针与数组
- ☑ 指针与函数
- ☑ 指针数组、指向指针的指针变量

指针是 C 语言一种重要的数据类型。在动态数据结构及其应用中，指针起着不可替代的作用。正确而灵活地运用指针，可以有效地表示复杂的数据结构；能动态地分配内存；能方便地使用字符串；有效而方便地使用数组；在调用函数时能得到多于 1 个的返回值；能直接处理内存地址；等等。可以说，不掌握指针就是没有掌握 C 的精华。

12.1　概述

12.1.1　存储单元与地址基本概念

计算机系统内部的存储器是一种按线性结构组织的存储空间。系统的内存以字节为单位编号。这些编号被称为地址。在机器语言中，对存储器中指定存储单元的访问是通过被访问单元的地址来实现的。地址是用二进制编码的数据，所以存储单元的地址也被称为地址码。存储单元的地址从 0 开始编码。最大地址码取决于中央处理器中地址码的位数。这一范围被称为中央处理器的地址空间。

程序运行后，系统根据存储对象的类型在内存中为变量分配存储空间。这样，每一个对象的存储空间都可以用一个地址码来表示，如 Turbo C 系统中，为 int 型变量分配两个连续的字节，为 float 型变量分配 4 个连续的字节。对于具有多个内存单元的数据，由于它们存储在连续的内存单元中，因此用这些连续单元中的第一个单元的地址码，即首地址来表示该数据的内存单元地址。

例如，下面变量定义的内存分配情况如图 12-1 所示。

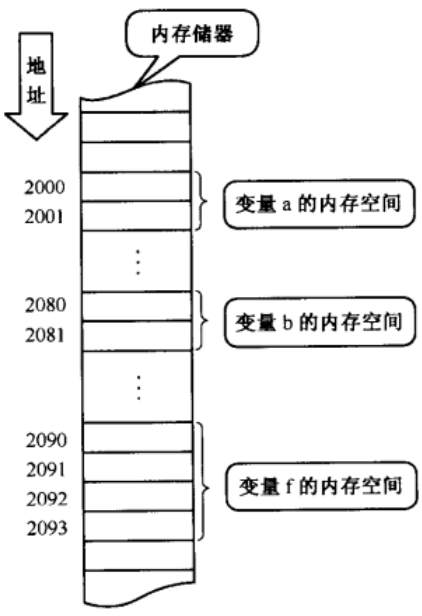


图 12-1　分配空间图

```
int a,b;
float f;
```

其中，变量 a 的地址编码为 2000，变量 b 的地址编码为 2080，它们都分别占两个连续的字节存储空间。而变量 f 的地址编码为 2090，占 4 个连续的字节存储空间。

所以，变量实际上就是存储单元。它是给存储单元所起的符号名称。对变量的访问，实质就是对存储单元的访问。

12.1.2　指针与指针变量的概念

指针：一个存储对象在内存中存放的起始地址被称为指向该对象的指针。

指针变量：保存其他对象的内存地址的数据对象，是指针类型的的变量。

通常，若 A 为一指针类型的数据，该数据为存储另一 B 数据的内存单元的首地址，则我们称指针 A 为指向 B 的指针，B 为指针 A 所指向的目标。这一关系可以用图 12-2 表示。

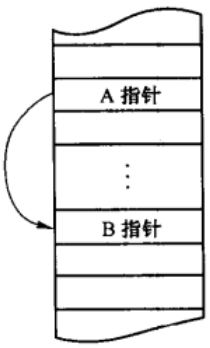


图 12-2　举例示意图

指针和指针变量提供了另一种访问变量的方法，即可以通过指针直接访问变量所在的内存空间。

注意： 指针是一个对象的地址指示器，而指针变量是一个数据对象，在这个指针变量中可以存放另一个对象的地址。

12.2　指针变量的使用

12.2.1　指针变量的定义与初始化

1. 指针变量的定义

C 语言规定，程序中使用的变量必须先定义，指定其类型。编译程序根据变量的类型分配内存单元。指针变量是用来存放地址的，必须定义成指针类型。

指针变量定义的格式为：

```
基类型　　*指针变量名
```

其中，“基类型”表示指针变量所指向的变量的类型，可以是我们前面所学的基本数据类型，也可以是后面将要学到的复合数据类型；“*”表示定义的是指针变量，而且要每个指针变量前面都要加上。

例如：

```
int   *p;                    /*其中 p 是一个指向整型变量的指针变量*/
char  s;                     /*其中 s 是一个指向字符变量的指针变量*/
long  *lp;                   /*其中 lp 是一个指向长整型变量的指针变量*/
int   *p1,*p2,*p3;           /*p1,p2,p3 是指向整型变量的指针变量*/
```

2. 指针变量的初始化

在定义指针变量的同时，还可以进行初始化，如：

```
int a;
int *p=&a;          /*将变量a的地址编码放到指针变量p中*/
```

当把一个变量的地址作为初始值赋给指针变量时，该变量必须在这个指针变量初始化之前定义过，且一般可用取地址运算符“&”将该变量的地址取出赋给指针变量。上面的初始化格式也可以写成如下的形式，即：

```
int a,*p=&a;
```

下面几种对指针变量初始化的做法是错误的。

```
float    *p=&b;
float    b;
```

在定义指针变量p时，变量b还没有定义，故不能获得变量b的地址。

```
int    a;
float *p=&a;
```

指针变量p的类型与变量a的类型不一致。注意，每一个指针变量只能指向一个同类型的变量。

```
float   *p=100;
```

不要将一个具体的数据赋给一个指针变量。这个例子的含义是将该数值作为一个内存地址，赋给指针变量p。如果对该地址进行写操作，则可能产生严重的错误。

12.2.2 指针的两个基本运算

1. 取地址运算符（&）

“&”运算用来计算存储某一数据的存储单元的首地址，而不是位运算中按位与运算符。

“&”运算的一般格式为：

```
& 左值
```

C语言中具有内存单元的数据被称为左值。左值可以是变量、数组元素及后面将要学的结构体、共用体中的成员等。它们的地址值都表示成变量名前加运算符“&”。

2. 间接访问运算符（*）

与“&”运算相对应的运算是“*”运算，用来对某一指针所指的目标进行访问，称其为间接访问运算。它只能用在指针类型的数据上。

“*”运算的一般格式为：

```
*指针表达式
```

例如，若有：

```
int *p,*q,x,y;
```

则可有：

```
p= &x;
q= &y;
* p=10;
* q= * p;
```

其中，语句“* p=10;”是将 10 赋给指针变量 p 所指的变量 x 中，而语句“* q= * p”是将指针变量 p 所指的变量 x 的值赋给指针变量 q 所指的变量 y 中。

间接访问运算表示对指针所指目标的引用。这里的引用包括对存储目标数据的内存单元的读和写两种操作。例如，上列中的计算机表达式“* q= * p”，“* q”表示对指针 q 所指目标，即 y 的写操作引用，而“* p”则表示对指针 p 所指目标 x 的读操作引用，有时在一个表达式中可能会有对同一目标同时进行两种操作引用的可能，如表达式：(* p) ++；表示将 p 所指向的目标读出加 1 后，再重新写入 p 所指向的目标。实际上，若某一指针指向某一变量，则由间接访问运算“*”同该指针构成的表达式完全等价目标变量名本身。

[例 12-1]　指针变量应用例子。

```
main()
{
    int   * p,a;
    char  * s,c;
    long * lp,d;
    a=5;
    d=351;
    c='f';
    p= &a;
    s= &c;
    lp= &d;
    printf("%d,%d,%c,%c,%ld,%ld",a, * p,c, * s,d, * lp);
}
```

其运算结果为：

```
5,  5,  f,  f,  351,351
```

[例 12-2]　交换两个指针本身的值。

```
main()
{ int a,b;
  int * p, * q, * t;
  a=100;b=10;
  p= &a;q= &b;
  printf("a= %d,b= %d \ n", * p, * q);
  t=p;
  p=q;
  q=t;
```

```
    printf("a=%d,b=%d\n", *p, *q);
}
```

其运行结果为：

```
a=100,b=10
a=10,b=100
```

3. 指针引用应注意的问题

一个指针变量只能指向一个同类型的变量。例如：

```
int *p;
```

定义了一个 int 类型的指针变量 p，则 p 只能指向 int 型的变量。或者说，p 中只能存放 int 型变量的地址。

表达式“*p”表示取得 p 所指向的对象。该对象必须是确定的。因此，引用一个指针变量之前，该指针必须已经指向一个变量。

运算符“&”和“*”的优先级相同，按自右向左的方向结合。如果有 p=&a，则有如下表达式的等价关系：

- p 等价于 &a，等价于 &*p；
- *p 等价于 a 等价于 *&a。

运算符“&”，“*”与运算符“++”，“--”的优先级相同，按自右向左的方向结合。因此，若有 p=&a，则有如下等价关系：

- (*p)++ 等价于 a++；
- *p++ 等价于 *(p++)。

注意区别以下两个关键的概念：

- 指针变量和指针变量指向的变量；
- 指针变量的值和指针变量所指向的变量的值。

12.3 指针与数组

一个数组在内存中占用一片连续的存储单元，数组名代表数组的首地址，因此数组名同指针是完全相同的数据类型。它们之间的运算就应该是通用的。

12.3.1 指向数组元素的指针变量与一维数组

用指针变量访问一维数组的各个元素的一般步骤是先定义一个指向数组元素的指针变量，然后通过赋值使指针变量指向某个数组元素后，就可能通过循环移动指针，引用数组的各个元素了。

[例 12-3] 分别用下标法和指针法访问一维数组。

程序 1：下标法。

```
main()
{ int a[5]={1,2,3,4,5},i;
  for (i=0;i<5;i++)
```

```
    printf("%5d",a[i]);
}
```

程序 2：地址法。

```
main()
{ int a[5]={1,2,3,4,5}, *p;
  for (p=a;p<a+5;p++)
    printf("%5d", *p);
}
```

程序 3：地址法（指针常量）。

```
main()
{ int a[5]={1,2,3,4,5},i;
  for (i=0;i<5;i++)
    printf("%5d", *(a+i));
}
```

程序中，定义 a 为整型数组，包含 5 个元素。数组元素在内存中的存放情况如图 12-3 所示。数组名 a 代表数组的首地址，而 a [0] 的地址 &a [0] 也是数组的首地址。因此，指针常量 a 与 &a [0] 等价。每个数组元素相当于一个变量，因此数组元素 a [i]（$i=0\sim4$）的地址与 &a [i] 或 a+i 二者等价。

程序 1 中使用下标法输出数组的各元素。实际上，在计算机内部处理时，将 a [i] 转变成 *(a+i) 后，计算 a+i，再访问与地址 a+i 对应的单元，如图 12-3 所示。

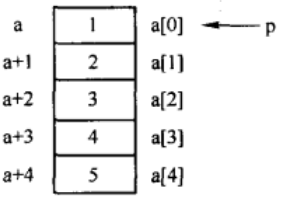


图 12-3　数组 a 存储图

程序 2 中使用指针变量访问数组的各元素。首先定义指向数组元素的指针变量 p，然后通过赋值语句 p=a，将数组 a 的首地址赋给指针变量 p，使 p 指向数组元素 a [0]。因此，该语句等价于“p= &a [0];”，然后通过循环使用间接访问符（*）输出 p 指向的数组元素的值，再通过“p++”依次指向下一个元素。

“p++”的含义是使指针变量 p 增 1，即指针 p 指向下一个数组元素，而不是下一个字节。在［例 12.3］中，“p++”使 p 的值实际增加了两个字节，因为 p 指向的数组元素是整型，一个元素占两个字节。

若指针变量 p 的初值为 &a [0]，则 p+i 和 a+i 都是 a [i] 的地址，所以 *(p+i) 和 *(a+i)都与 a [i] 等价，也可以写成 p [i]。但 p+10 是无效地址，因为它超出了数组 a 的范围。

可见，指针变量与整数进行加、减运算的结果只在数组的范围内有效，超出该范围会出错，但编译系统并不报错，属于隐含错误。

比较以上 3 个程序可以看出，程序 1 和程序 3 的执行效率是相同的，程序 2 比较快，因为 p++ 比计算 a+i 要快得多。但是，用下标法更直观，如 a [3] 是数组中下标为 3 的元素，而程序 2 中需要弄清指针变量 p 的当前指向，才能知道输出的是哪个元素。

在使用时，对本函数内定义的数组使用下标法更加方便；对于通过参数传递的数组要用

指针变量作为形参，通常用指针法访问数组。

注意：C语言规定，如果设p是指向数组元素的指针，则不管该数组是什么类型，p+1是下一个数组元素的指针，此时的“1”是一个数组元素，而不是一个字节，因此经常在循环中用“p++”使指针变量p依次指向下一个数组元素。同理，“p--”表示p指向前一个数组元素；p+n表示指针p指向的元素后面第 n 个元素的地址。具体移动的字节数由指针指向的数组元素的类型决定。

[例12-4] 编写函数，实现将数组中的n个整数逆序存放。要求：在主函数中调用该函数，实现将10个整数逆序存放。

在主函数中，设10个整数用整型一维数组a表示，为了使运行结果更清楚，逆序前后分别输出数组a的各元素。

函数inv（）用于实现逆序。由于该函数不需要通过返回值实现数据传递，因此函数类型定义为空类型。函数间要传递的数据是一个数组a，需要有两个参数，即数组的首地址和元素个数，这样才能惟一确定一个数组。因此，形参也有两个：一个是指向数组元素的指针变量q，用来接收数组a的首地址；另一个是整型变量n，用来接收实参数组中元素的个数。

函数体的实现：逆序时，将数组中头尾对应位置上的元素互换，即a[0]与a[9]交换，a[1]与a[8]互换，a[2]与a[7]互换，依此类推，直到数组中间的数据互换完为止。本函数用指针法访问数组元素。设指针变量i，j开始时分别指向a[0]和a[9]，则循环执行*i与*j互换，用i++使i指向下一个元素，用j--使j指向前一个元素。循环条件为i<j。

其程序如下：

```
main()
{int  a[10]={1,2,3,5,7,9,12,23,4,123},i;
 int *p;
void  inv(int  *q,int n);
  printf("\n before  inverse :\n");
  for(p=a;p<a+10;p++)
      printf("%6d",*p);
inv(a,10);
for(i=0;i<10;i++)
  printf("%6d",a+i);
}
void  inv(int *q,int n)
{ int *i,*j,t;
  for (i=q,j=q+n-1;i<j;i++,j--)
       t=*i,*i=*j,*j=t;
}
```

12.3.2 指针与二维数组

同一维数组类似，二维数组也可通过指针变量来访问，但用指针访问二维数组及多维数组较复杂一些。

在二维数组中，整个数组有一个首地址，即二维数组的指针。数组中的每一行也有一个首地址，通常被称为行指针。每个数组元素中也有一个具体的地址，即指向数组元素的指针。

首先回顾一下二维数组的性质。例如，有如下 3 行、4 列的二维数组 a：

```
int a[3][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12}};
```

在第 10 章中曾经介绍过，二维数组在内存中是按行存放的，即从低地址到高地址分别存放数组元素 a [0] [0] ～a [1] [3]，a [1] [0] ～a [1] [3]，a [2] [0] ～a [2] [3]。

首先，分析数组元素的存放情况：数组 a 占用连续的 12 个单元（每个单元存放一个整数），数组元素按行依次排列，与一维整型数组 b（设定义：int b [12];）的存储结构完全相同，只不过数组元素的名字不同而已。

其次，分析二维数组中的指针：a 是数组名，代表二维数组的首地址。a 数组的每一行又是一个一维数组（数组名分别是 a [0]，a [1]，a [2]），a 数组可以看作包含 3 个数组元素的一维数组，所以 a，a+1，a+2 分别是第一行、第二行、第三行的首地址（即 a [0]，a [1]，a [2]），即称其为行指针，又称其为指向一维数组的指针。每个数组元素的地址为指向数组元素的指针，如 &a [0] [0]，&a [2] [0] 等。

根据二维数组的存储结构，可以将二维数组看作两种特殊的一维数组。

将二维数组看作一个一维数组，使用指向数组元素的指针变量访问每一个数组元素。

数组 a 可以看作长度为 12 的整型一维数组，第一个数组元素的地址为 a [0] 或 &a [0] [0]。

将二维数组看作一个特殊的一维数组，每个数组元素又是一个一维数组，使用指向一维数组的指针变量访问二维数组。

下面通过应用实例比较两种指针的使用。

[例 12-5] 利用指向数组的指针访问二维数组的各元素输出整型二维数组 ss（2 行 3 列）的各元素。

首先将数组 ss 看作包含 6 个数组元素的一维数组，定义 p 为指向数组元素的指针变量，将数组的第一个数组元素（ss [0] [0]）地址赋给 p，通过循环移动指针 p 访问每一个数组元素。

其程序如下：

```
main()
{  int  ss[2][3]={2,4,6,7,8,10};
   int *p;
   printf("\n");
   for(p=ss[0];p<ss[0]+6;p++)          /*p=ss[0]与 p=&ss[0][0]等价 */
   printf(("%6d", *p);
}
```

注意： 因为 p 是指向整型数据（数组元素）的指针变量，因此初值必须是第一个数组元素的地址 ss [0] 或 &ss [0] [0]，虽然指针 ss 也代表数组的首地址，其值与 ss [0] 相等，但 ss 是指向一维数组的指针，而 ss [0] 是指向数组元素的指针。“p++”的用途是使 p 指向下一个数组元素。

[例 12-6] 利用指向一维数组的指针变量访问二维数组。利用指向一维数组的指针变量 p 输出数组 ss [2] [3] 中的各元素。

首先定义指向一行的指针变量。其一般格式为：

```
类型说明符  (*变量名)[n]
```

其中，类型说明符是数组元素的类型；n 是二维数组的列数，即指针变量所指向的一维数组的长度。需要特别强调的是，圆括号不能省略。

例如：

```
int (*p)[3];
```

定义指向长度为 3 的整型一维数组的指针变量 p。也就是说，p 只能指向包含 3 个整型元素的一维数组。

有了行指针变量后，将数组 ss 的首地址（ss 或 &ss [0]）赋给 p，使得 p 指向第一行。此时的 * p 表示 ss [0]，* ((* P) +1)、(* p) [1]、p [0] [1] 均表示 ss [0] [1]，* ((* p) +2)表示 ss [0] [2]，依此类推。

因此，利用 p 访问数组元素的方法是先使 p 指向二维数组的第一行，再通过“p++”使 p 指向下一行（即移动一个长度为 3 的整型一维数组的字节数），p 指向的一行中，列下标为 j 的元素为 * ((* p) +j) 或 (* p) [j]。

通过 p 访问数组 ss 的程序如下：

```
main()
{ int  ss[2][3]={2,4,5,6,7,8};
  int (*p)[3],j;
  printf("\n");
  for(p=ss;p<ss+2;p++)
    for (j=0;j<3;j++)
      printf("%6d",*((*p)+j));
}
```

通过以上两例可以看出两种指针变量的使用区别：指向数组元素的指针变量可以访问任意长度的数组，指向一维数组的指针变量只能访问固定长度的数组。

因此在实际应用中，当需要用指针变量作为函数参数，编写处理任意二维数组的数据时，采用指向数组元素的指针变量访问二维数组更合适。

[例 12-7] 编写函数，实现求 $m \times n$ 矩阵（整型）中的最大值元素及下标。要求编写主函数调用该函数，并在主函数中输出结果。

因为函数要求能处理任意大小的矩阵，所以不能用指向一维数组的指针变量，而采用指向数组元素的指针变量访问矩阵对应的二维数组。

设函数名为 max，则函数类型可定义为 void，形参有 5 个，指向数组元素的指针变量 p 接收数组中第一个元素的地址，整型变量 m 和 n 接收主函数中二维数组的行数和列数，指向变量的指针变量 r 和 c 接收主调函数中最大值元素的行和列下标变量的地址。

算法提示：设变量 i，j 分别表示数组元素的行、列下标，变量 kr，kc 分别表示最大值

元素的行、列下标，则用双重循环访问每一个数组元素。

关键问题：若 p 指向数组元素 ss [0] [0]，则 ss [i] [j] 的等价指针法表示为 *(p+i*n+j)，下标法表示为 p [i*n+j]。如在 4 行、3 列的数组中，ss [2] [1] 的地址为 p+2*3+1，对应的元素为 * (p+2*3+1) 或 p [2*3+1]。

其程序如下：

```
main()
{  int  ss[4][3]={1,12,3,14,5,16,7,8,9,2,13,4};
int max_r,max_c;
max(ss[0],4,3,&max_r,&max_c);
printf("\n max:%d,row:%d,clum:%d",ss[max_r][max_c],max_r,max_c);
}
void  max(int  *p,  int m,int n,int *r,int *c)
{ int i,j,kr,kc;
  kr=kc=0;
  for (i=0;i<m;i++)
    for(j=0;j<n;j++)
      if(p[i*n+j]>p[kr*n+kc])  {kr=i;kc=j;}
*r=kr, *c=kc                          /*通过 r、c 访问它所指向的变量 max_r,max_c */
}
```

其运行结果为：

```
max:16,row:1,colum:2
```

在函数 max () 中还可以通过移动指针 p 访问每一个数组元素。其对应的程序段如下：

```
void  max(int  *p,int m,int n,int *r,int *c)
{ int  i,j,kr,kc;
  int max1=*p;
  kr=kc=0;
  for(i=0;i<m;i++)
    for(j=0;j<n;j++,p++)
     if(*(p)>max1) {max1=*p,kr=i,kc=j;}
  *r=kr, *c=kc;
}
```

12.3.3　指针与字符串

在 C 语言中，字符串都是用字符数组保存的。因此，对字符串的访问也有两种，即下标法和指针法。

(1) 下标法

下标法用字符数组存放一个字符串，然后输出该字符串。

[例 12-8]　定义一个字符数组，对它初始化，然后输出该字符串。

```
main()
{char str[ ]="i love China!";
printf("\n%s",str);
}
```

其运行结果为：

```
I  love  China!
```

(2) 指针法

前面学习过字符串存储时以字符串结束标志‘\0’结束，因此访问字符串时，只要有了字符串首地址（字符数组名或指向字符串首字符的指针），就能惟一确定一个字符串，不需要给出字符串的长度。编程时经常以‘\0’作为字符串的结束标志。

那么，如何使用字符指针访问字符串呢?

与同一维数组的指针访问法类似，先定义指向字符的指针变量，然后通过赋值语句使该指针指向字符串的首地址，再通过该指针访问字符串。

[例 12-9] 输入任一字符串，通过字符数组名和字符指针分别输出该字符串。

设程序中用一维字符数组 string 保存输入的字符串，用指向字符的指针变量 str 访问字符串。使用指针访问字符串之前，先将字符串的首地址赋给指针变量。

其程序如下：

```
#include  <stdio.h>
main()
{  char  string[80], *str=string;
   gets(string);
   puts(sting);
   puts(str);
}
```

其运行结果为：

```
Thank you!
Thank you!
Thank you!
```

可见，利用字符数组名与利用指针变量访问字符串效果相同。实际上，字符数组名也是指针，是指向字符串的指针常量。

12.4 指针与函数

一个函数在内存中占据一段连续的存储单元。这些存储单元的首地址就是函数的指针。通常，函数名代表函数的首地址，因此可以定义一个指针变量来存放函数的首地址，使指针变量指向函数，然后通过该指针变量实现函数的调用。

函数的返回值可以是整型、实型、字符型等，也可以是指针类型，即返回一个地址值。

因此，函数与指针的综合应用主要体现在以下两个方面：

- 用指向函数的指针变量调用函数；
- 编写返回指针值的函数，常用于字符串处理。

12.4.1　指向函数的指针变量

指向函数的指针变量用于保存函数的首地址。其具体使用步骤与指向变量、数组等指针变量类似，即先定义指向函数的指针变量，通过赋值使指针变量指向函数，然后通过指针调用函数。

指向函数指针变量的一般定义形式为

```
类型标识符号 (*指针变量名)([参数类型])
```

其中，类型标识符号是指函数返回值的类型；参数类型是定义函数时的形参类型，可以省略，这时可以匹配任意的实参表，但有时会出错。

通过赋值语句使指针变量指向某函数，就可以用指针变量间接调用该函数了。其调用格式为

```
(*指针变量名)(实参)
```

在实际应用中，指向函数的指针变量主要用来编写通用函数。具体地说，就是用指向函数的指针变量作为函数形参，接收实参地址，即用函数名作为实参，然后间接调用实参函数。

[例 12-10]　编写函数 integrate ()，求任意函数 $f(x)$ 在区间 (a, b) 上的定积分。

函数 integrat () 返回计算后的定积分值，因此函数的类型就为实型，参数应该有 3 个，包括函数 $f(x)$、区间值 a 和 b。其中，要传递函数必须用指向函数的指针作为形参，用函数名作为实参，以传递函数的首地址。

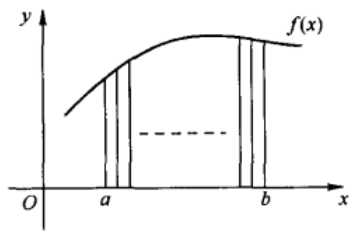


图 12-4　求积分示意图

求定积分的方法采用梯形法，如图 12-4 所示，即将曲线 $f(x)$ 与 x 轴、直线 $x=a$、直线 $x=b$ 围成的多边形划分成 n 等份，每一份近似地看作是一个小梯形，如最左边的小梯形面积是 $(f(a)+f(a+h))*h/2$。将所有小梯形的面积相加，即可得到定积分值。将其化简后的公式为

$$h\left[\frac{f(a)+f(b)}{2}+f(a+h)+f(a+2h)+\cdots+f(a+(n-1)h)\right]$$

其中，n 为将区间 a、b 划分的等份数，$h=\dfrac{b-a}{n}$。

其程序如下：

```
double  integrate(double  (*f)(float), double  a,double b)
{double  s,h,y;
  int n,i
  n=100;
  h=(b-a)/n;
  s=((*f)(a)+(*f)(b))/2;
```

```
    for(i=1;i<n;i++)
      s=s+(*f)(a+i*h);
    y=s*h;
    return(y)
}
```

以后就可以调用上述函数求任意函数 $f(x)$ 在给定区间上的定积分了。例如，求以下函数的定积分，即

$$y_1=\int_0^2(x^2+1)\mathrm{d}x$$

$$y_2=\int_1^{3.5}(2x-1)\mathrm{d}x$$

编写函数 f_1 ()、f_2 () 分别实现：$f_1(x)=x^2+1$，$f_2(x)=2x-1$，最后在 main () 函数中调用定积分函数 integratr ()，每次调用时求一个函数 $f(x)$ 在给定区间上的定积分。

其程序如下：

```
double f1(float x)
{return(x*x+1);}
double f2(float x)
{return(2*x+1);}
main()
{double f1(float),f2(float),integrate(double (*f)(float),double,double)
  double y1,y2;
  y1=integeatr(f1,0.0,2.0);
  y2=integrate(f2,1.0,3.5);
  printf("\n y1=%6.2f,y2=%6.2f",y1,y2);
}
```

其运行结果为：

```
y1=4.67,y2=8.75
```

程序中，为了保证程序的正确性，函数声明时最好写明形参的类型。

注意：

- (*f) (float) 表示定义一个指向函数的指针变量。它不是固定指向哪个函数的，而只表示定义了这样一个类型的变量。它是专门用来存放函数的入口地址的。在程序中，把哪一个函数的地址赋给它，它就指向哪一个函数。在一个程序中，一个指针变量可以先后指向返回类型相同的不同的函数。
- 在给函数指针变量赋值时，只需给出函数名而不必给出参数。
- 用函数指针变量调用函数时，只需将 (*f) 代替函数名即可 (f 为指针变量名)，在 (*f) 之后的括号中根据需要写上实参。
- 对指向函数的指针变量，像 p++，q++ 等运算是无意义的。

12.4.2　返回指针值的函数

返回指针值的函数与前面介绍过的函数的不同之处是函数的返回值是指针类型。其一般定义形式为：

```
类型标识符号　　*函数名(形参表)
```

如：

```
float　*max(float　x,float　y){　}
```

定义一个函数 max（），函数的返回值为指向实型数据的指针，即函数的类型是float *。

字符串处理函数经常需要返回地址，如查找某个字符在字符串中第一次出现的位置，两个字符串连接成一个字符串等。这时用返回指针值函数比较方便。

[例 12-11]　编写函数 search（），查找字符 ch 在字符串 ss 中第一次出现的位置。在主函数中输入字符串及要查找的字符，输出查找结果。

函数 search（）的类型为指向字符数据的指针类型，参数有两个，即字符型变量 ch 和指向字符型的指针变量 str。

算法：从 str 指向的字符串的首字符开始，循环比较 str 指向的字符是否与字符 ch 相等。若相等，则结束循环，返回指向该字符的指针；否则，str 指向下一个字符，直到字符串结束标志‘\0’为止，返回空指针（NULL），表示没找到。

其程序如下：

```
main()
{ char　*search(char　*str ,char　ch);
  char　ss[80], *p,chr;
  printf("\nintput a char for search:");
  ss=gets();
  chr =getchar();
  p=search(ss,chr);
  if(p==NULL)
    printf("\n　not　found");
  else
    printf("\n position:%d (count　from　0)",p-ss);
}
char　*search(char　*str,char　ch)
{ while(*str!　='\0')
    if(*str==ch)return(str);
    else　str++
  return(NULL);
}
```

12.5　指针数组与指向指针的指针变量

本节内容是关于指针的较复杂的应用。初学者可先理解基本概念，以后用到时再深入学

习其应用。

12.5.1 指针数组

通过前面的程序可以看出，一个指针变量只能存放一个地址。如果要同时保存多个指针（或地址），就需要多个指针变量。多个指针变量组合在一起，就是指针数组。指针数组实际上是指向相同类型数据的指针变量的集合。每个数组元素相当于一个指针变量。

为什么要用指针数组呢？其主要原因有三个：一是用指针数组处理多个字符串很方便；二是用指针数组可以传递任意大小的二维数组；三是用指针数组作为 main（_）函数形参可以向程序传送命令行参数。

1. 用指针数组处理多个字符串

假如要处理一些书名，每个书名是一字符串，而且长度各不相同。如果用二维数组表示，则每行保存一个字符串，各行长度相同，当某个字符串较短时，则容易造成存储空间的浪费，如图 12-5 所示。而用指针数组表示，则每个数组元素指向一个字符串，各字符串占用的空间与字符串的实际长度有关。相比而言，用指针数组表示更节省内存，如图 12-6 所示。

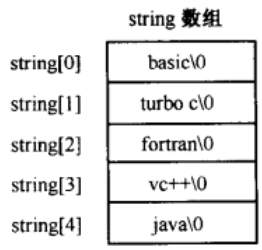


图 12-5 二维字符数组存储字符串

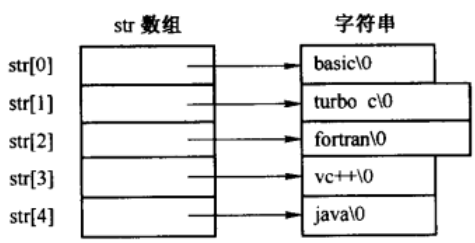


图 12-6 指针数组存储字符串

[例 12-12] 使用指针数组访问如图 12-6 所示中给定的 5 个字符串。

其程序如下：

```
#include <stdio.h>
main()
{ char *str[]={"basic","turbo c","fortran","vc++","java"};
  int i;
  for (i=0;i<5;i++)
  puts(str[i]);
}
```

程序中，定义指向字符指针数组 str 并初始化，使每个数组元素存放一个字符串的首地址（即每个元素指向一个字符串），如图 12-6 所示，然后通过指针数组输出所有的字符串。

可见，一维指针数组的定义与基本类型的数组相同，只不过数组元素的类型不同而已。其一般形式为：

```
类型说明符        *数组名[数组长度]
```

如 char *str [5];

表示定义指针数组 str，包含 5 个数组元素，每个元素的类型是字符指针类型。

注意：指针数组与指向数组的指针变量的区别，如。

```
char (*p)[7];
```

其含义是定义指向长度为 7 的一维字符数组的指针变量 p。

假如要对如图 12-6 所示的书名排序，如果用二维数组表示，则交换时需要移动整个字符串；而用指针数组表示，则排序时不必移动字符串位置，只要改变指针数组中各元素的指针即可，如图 12-7 所示。这样，交换指针变量的值比交换两个字符串要快得多。因此，用指针数组可以实现字符串的快速排序。

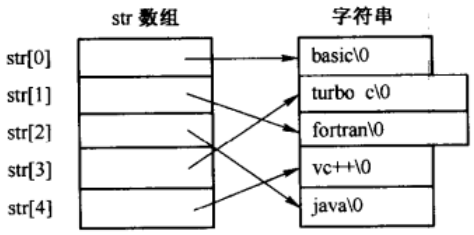


图 12-7　指向字符串指针排序

[例 12-13]　将字符串按字母顺序排序。

其程序如下：

```
#include <stdio.h>
main()
{
void sort(),prn();
  char *str[]={"basic","turbo c","fortran","vc++","java"};
  sort(str,5)
  prn(str,5);
}
void sort(char *v[],int m)
{
  int i,j,k;
  char *t;
  for(i=0;i<m-1;i++)
    {k=i;
      for(j=i;j<m;j++)
        if (strcmp(v[k],v[j])>0) k=j;
      if (k!=i)
          {t=v[k];v[k]=v[i];v[i]=t;}
    }
}
void prn(char *v[],int m)
{ int t;
  for(i=0;i<m;i++)
    puts(v[i]);
}
```

其运行结果为：

```
basic
fortram
java
turbo c
VC ++
```

2. 用指针数组处理任意大小的二维数组

[例 12-14] 编写函数，求任意大小的矩阵中各行元素之和。在主函数中调用该函数求2×3矩阵中各行元素之和，并输出结果。

其程序如下：

```
void  aver(int  *p[],int  m,int n,int *s)
{ int  i,j;
  for(i=0;i<m;i++)
    {s[i]=0;
     for(j=0;j<n;j++)
       s[i]+=p[i][j];
  }
}
main()
{ int a[2][3]={5,6,7,8,1,4}, *c[2],b[2];
  c[0]=a[0];c[1]=a[1];
  aver(c,2,3,b);
  printf("\nsum  of  line 1:%d,line 2%d",b[0],b[1]);
}
```

其运行结果为：

```
sum of line 1:18     line 2:13
```

调用函数 aver（）时，将指针数组 c 的首地址传给形参 p，即 p 指向数组 c 的第一个元素 c［0］，将数组 b 的首地址传给形参 s，将行、列数组传给 m、n。在函数 aver（）中，通过 p 引用数组 c，再通过 c 引用 a，通过 s 访问 b。其中，p［i］等价于 *（p+i），即第 i 行的首地址 c［i］。因此，p［i］［j］是第 i 行、第 j 列的元素。

可见，用指针数组作为函数参数可以处理任意大小的二维数组。此时，指针数组的每个元素是一个指向数组元素的指针，指向每一行的第一个元素，相当于一维数组名。

12.5.2 指向指针的指针变量

指针变量也有地址，如果将该地址用变量存放，则该变量是指向指针的指针变量。如定义以下变量，即：

```
int  i=100, *p, * *q;
```

则 q 是指向指针的指针变量。执行以下赋值语句：

```
p= &i;
q= &p;
```

执行后各变量之间的关系如图 12-8 所示。

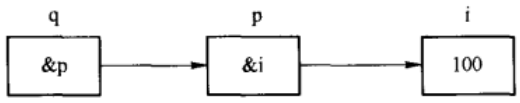


图 12-8　各变量之间的关系

以后，要访问变量 i（如赋值），可以使用以下三种方法：

```
i=10;          /* 直接访问 */
*p=10;         /* 一次间接访问 */
**q=10;        /* 二次间接访问，*q 是指向整型的指针 p */
```

C 语言允许进行多重间接访问，也就是说，定义变量时可以有多个 *，不过极少使用两次以上的间接访问。对于初学者来说，两次已经够复杂的了。

指向指针的指针主要用来操作指针数组，例 12-13 中的 prn（）函数可以用指向指针的指针作为参数，然后用指向指针的指针操作字符串。

```
void  prn(  char  **v,int  m)
{int i;
    for(i=0;i<m;i++)
      puts(*v++);
}
```

调用函数 prn（）时，形参 v 接收实参数组 str 的首地址，即指向数组元素 str［0］。v++ 使指针指向下一个指针数组元素，由此间接访问指针数组各元素指向的字符串。

12.5.3　用指针数组作为 main（）函数的参数

指针数组的另一个重要应用是用指针数组名（或指向指针的指针变量）作为 main（）函数的形参，从命令行接收参数。

在前面的程序中，main（）函数都有没有参数，实际上 main（）函数也可以带参数。C 语言规定，main（）函数可以带两个参数。其函数原型为：

```
main(int argc,char  *argv[])
```

C 语言规定，C 程序从 main（）函数开始执行，那么 main（）函数的实参从哪来呢？

实际上，除了在 TC 集成环境中运行 C 语言程序外，经编译连接生成可执行文件的 C 程序可以在任何操作命令状态下直接运行。当处于操作命令状态下（如在 MS－DOS 中的命令状态 C:\TC>），输入程序名及参数（即命令行），系统就调用程序中的 main（）函数开始执行，同时将参数传给 main（）函数的形参。

命令行的一般形式为：

```
程序文件名    参数 1    参数 2    参数 3 ……参数 n
```

其中，程序文件名和各参数之间以空格分隔。通常我们说命令行参数的个数也包含程序文件名在内。完整的程序文件名应包括路径、文件的主名和扩展名，如 d:\tc\user\ch10_4.exe，表示 d 盘 tc 文件夹中 user 目录中的程序文件 ch10_4.exe。运行时，扩展名可省略。

main（）函数的形参与命令行实参的对应关系为整型变量 argc 用来接收命令行参数个数，每个参数看作一个字符串，指向字符的指针数组 argv 用来接收命令行各参数字符串的首地址。

[例 12-15] 将命令行参数输出（设程序文件名为 mainfile.exe，存于 d:\tc\user 目录下）。

其程序如下：

```
#include <stdio.h>
main()
{   int i;
    for (i=0;i<argc;i++)
    puts(argv[i]);
}
```

在 DOS 命令行状态下运行（下划线部分为用户输入）：

```
d:\tc\user>mainfile f1 f2
D:\TC\USER\MAINFILE.EXE
f1
f2
```

程序运行时，命令行有 3 个参数，即 mainfile、f1、f2。因此，argc 的值为 3。3 个参数字符串首地址分别传给指针数组 argv 的 3 个数组元素，即 argv［0］指向字符串“mainfile”，argv［1］指向字符串“f1”，argv［2］指向字符串“f2”。

形参数组名 argv 相当于指向数组元素的指针变量，因此以上程序可等价于

```
#include <stdio.h>
main(int argc,char **argv)
{ while(argc-->0)
  puts(*argv++);
}
```

程序中，argv 开始时指向数组的第一个元素，*argv 是第一个参数字符串的首地址，argv++ 使得 argv 指向下一个字符串的首地址。

注意：利用指针数组作为 main（）函数参数，可以向程序传送命令行参数，而且参数的个数任意，参数字符串的长度也各不相同。但参数中不能含空格，因为空格可作为参数之间的分隔符。

小结

指针是 C 语言中一个重要的数据类型，也是 C 语言的一个重要特色。在程序中，灵活

地应用指针，可以使程序简捷、紧奏、高效。

本章主要介绍了指针的概念和指向不同类型数据的指针的应用。

在指针的基本概念部分，主要讲述了指针是专门表示地址的一种数据类型，如何定义一个指针变量，取地址运算符（&）和间接访问运算符（*）。重点掌握如何定义一个指针变量和指针的两个运算符。

指向不同类型数据的指针变量的应用主要包括以下几个方面。

- 通过指向变量的指针变量可以间接访问变量，用该指针变量作为函数形参，可实现在被调函数中改变主调函数中变量的值，解决函数间需要返回一个以上的返回值的问题。
- 通过指向数组元素的指针变量和指向数组的指针变量均可以访问数组，用该指针变量作为函数形参，可以实现主、被调函数共用一个数组，即在被调函数中访问主调函数中的数组。
- 通过指向函数的指针变量可以间接调用函数，用该指针变量作为函数形参可实现通用函数的编写。
- 函数的返回值也可以是指针，通常在处理字符串函数中使用。
- 通过指针数组可以很方便地访问多个字符串，还可以作为 main（）的参数以便接收命令行参数。
- 指向指针的指针变量可以实现多级间接访问。

习题

1. 若有以下语句 int a [10]，*p=a，k;，且 k 在合理的范围内，那么对数组元素引用错误的是_____。

A. *（a+k）　　B.a [p-a]　　C.p+k　　D. *（&a [k]）

2. 在下列语句中，正确的是_____。

A.char a [3] [] = {'abc','i'};　　B.char a [] [3] = {'abc','i'};

C.char a [3] [] = {'a',"i"};　　D.char a [3] [] = {"a","i"};

3. 以下定义和初始化的两个数组 a1 和 a2，那么下列说法正确的是_____。

```
char  a1 [] ="program";
char  a2 [] = {'p','r','o','g','r','a','m'};
```

A.a1 和 a2 完全相同

B. a1 和 a2 不同，a1 是指针

C. a1 和 a2 存储单元的数目相同

D. a1 和 a2 不同，a1 的存储单元数目多

4. 若有定义"char a [10]，*p=a;"，那么下列赋值语句正确的是_____。

A. a [] ="abcdefg";　　B. a="abcdefg";

C. *p="abcdefg";　　D. p="abcdefg";

5. 下面程序段的输出结果是_____。

```
#include  <stdio.h>
```

```
main()
{int a[]={1,2,3,4,5, 6}, *p;
p=a;
*(p+3)+=2;
printf("%d,%d", *p, *(p+3));
}
```

A. 0, 4　　B.1, 4　　C. 0, 6　　D. 1, 6

6. 若有定义：

```
int c[4][5],(*pc)[5];
pc=c;
```

那么，下列对数组 C 的元素引用正确的是_____。

A. pc+1　　B. *(pc+3)　　C. *(pc+1)+3　　D. *(*pc+2)

7. 定义语句"double *array [8]"的含义是_____。

A. array 是一个指针，它指向一个数组，数组的元素是双精度实型

B. array 是一个数组，数组的每一个元素是指向双精度实型数据的指针

C. C 语言中不允许这样的定义语句

D. 以上都不对

8. 请写出下列程序的输出结果。

(1)
```
main()
{int a[5]={2,4,6,8,10}.*p,**k;
p=a;
k=&p;
printf("%d,",*(p++));
printf("%d",**k);
}
```

(2)
```
main()
{static int a[][3]={9,7,5,3,1,2,4,6,8};
int i,j,s1=0,s2=0;
for(i=0;i<3;i++)
for(j=0;j<3;j++)
{ if (i==j)s1=s1+a[i][j];
if(i+j==2)s2=s2+a[i][j];
}
printf("%d\n%d\n",s1,s2);
}
```

(3)
```
main()
 {char b[]="abcdefg";
  char *pc=&b[7];
  while(--pc>b)
    putchar(*pc);
```

```
    putchar('\n');
}
```

以下各题用指针方法编程。

9. 输入 3 个整数，按由小到大的顺序输出。

10. 输入 10 个整数，将其中最小的数与第一个数对换，把最大的数与最后一个数对换，写 3 个函数：

(1) 输入 10 个数；

(2) 进行处理；

(3) 输出 10 个数。

11. 统计一行字符中大写字母、小写字母、数字及其他字符的个数。在主函数中输入一行字符，调用该函数进行统计，在主函数中输出统计结果。

12. 编写函数 char * fun (char * s, int n) 实现：将字符串 s1 中的字符“循环左移” n 位。例如，输入“ABCDE”，循环左移两位后应输出“CDEAB”。

13. 有 n 个人围成一圈，从 1 开始顺序排号。从第一个人开始报数（从 1～3 报数），凡报到 3 的人退出圈子，问最后留下的是原来的第几号？

提示：可以用数组保存每个人的序号，当退出时，该序号置为 0，同时计数变量 n 减 1。当 n 等于 1，即只剩下一个人时，序号为非 0 的就是所求结果。报数的过程中，用指针依次访问各数组元素。

14. 用指针数组实现：输入一月份号（1～12），输出该月份对应的英文名称。如输入“3”，则输出“March”。

15. 用函数实现二维数组的转置。

16. 在主函数中输入 6 个字符串，调用另一个函数对这些字符串进行排序（按字典顺序从小到大），然后在主函数中输出排序后的字符。

第 13 章　结构体与共用体

本章要点

- ☑ 结构体
- ☑ 结构体数组
- ☑ 指向结构体类型数据的指针
- ☑ 共用体
- ☑ 枚举类型
- ☑ 用户自定义类型

13.1　结构体

结构体是将一些不同的数据类型组织在一起而形成的一种数据类型，属于一种构造类型。因此，一个结构体可以有若干个数据项，每一个数据项的数据类型可以不同。结构体中的数据项又被称为分量、成员或属性。

13.1.1　定义结构体类型的一般形式

格式如下：

```
struct 结构体名
    {成员 1;
     ……
     成员 n;
    };
```

每个成员的定义格式：

```
类型标识符    成员名
```

其中，struct 是关键字，结构体名与成员名的命名规定同标识符的规定。

注意： 在定义的最后（即大括号“}”之后）的分号不能默认。

例如：

```
struct teacher
  { int   number;
    char  name[20];
```

```
    char  sex;
    char  profession[6];
    char addr[30];
};
```

该例定义了一个结构体类型。其中，struct teacher 为结构体名；大括号括起来的内容为该结构体类型包含的各个数据项（即成员）。teacher 结构包含的成员有 number、name、sex、age、profession 和 addr。它们的类型在定义时一起指定。但要注意，成员名不是一个变量名。另外，结构体类型的定义可以在函数的内部，也可以在函数的外部。在函数内部定义的结构体，只在函数内部可见。在外部定义的结构体，从定义点到源文件尾之间的所有函数中都有可见。当结构体类型 struct teacher 定义好后，就可能用这种 struct teacher 类型（像前面所学的 int、float、char 等数据类型一样）去定义结构体变量。

13.1.2　结构体变量的定义

结构体变量有三种定义方法：

- 先定义结构体类型再定义变量；
- 在定义类型的同时定义变量；
- 直接定义结构体类型变量。

(1) 先定义结构体类型再定义结构体变量

例如：

```
struct  teacher
  { int  number;
    char  name[20];
    char  sex;
    char  profession[6];
    char addr[30];
};
......
struct teacher  t1,t2;
```

上面先定义了一个结构体类型，然后用它定义了两个结构类型变量 t1 和 t2。变量 t1 和 t2 都具有 struct teacher 类型的结构。由于 t1 和 t2 都是变量，故计算机系统将为变量 t1 和 t2 分配必要的存储空间（一个结构体变量在内存中所占字节数是其所有成员变量占用的字节数之和），而结构体类型 struct teacher 并不分配存储空间。它仅仅是定义了一种类型说明，不需要存储空间。注意结构体类型和结构体变量的区别。结构体类型（如 struct teacher）就像 int、float、char 等数据类型一样，是一种类型说明符。

(2) 定义结构体类型的同时定义结构体变量

定义的形式：

```
strct 结构体名
    {成员 1;
```

```
    ……
    成员 n;
}变量名表列;
```

例如：

```
struct  teacher
{ int  number;
  char  name[20];
  char  sex;
  char  profession[6];
  char addr[30];
}t1,t2;
```

该例的作用与上面的例子相同，也是定义了一个结构体类型 struct teacher，同时也定义了两个结构体类型变量 t1 和 t2。

(3) 直接定义结构体类型变量

定义的形式：

```
struct
{ 成员 1;
  ……
  成员 n;
}变量名表列;
```

例如：

```
struct
{ int  number;
  char  name[20];
  char  sex;
  char  profession[6];
  char addr[30];
}t1,t2;
```

该例是定义了两个结构体类型变量 t1 和 t2。用这种方法定义，由于不出现结构体类型名，因而只适用于对变量进行一次性定义的情况。

上面介绍了结构体类型变量的 3 种定义方法。其中，第一种方法是最常用的方法。在定义结构体类型和结构体变量时应注意：先定义类型，后定义变量。变量分配必要的存储空间，而类型不分配存储空间。结构体类型成员名可以与程序中的变量名相同，二者代表不同的对象。成员既可以是一个基本类型（如 int，float，char 等），也可以是一个构造类型（如数据组、结构体等），因而结构体可以嵌套，即一个结构体类型的成员是另一种结构体类型的变量。

例如，在上述 teacher 结构体中增加了一个“出生日期”成员 birth。该成员需要包括年、月、日 3 个属性，因此可先定义一个 date 结构体类型，并用该类型定义 birth 成员。

其程序如下：

```
struct  date
  { int  year;
    int  month;
    int  day;
  };
struct teacher
  { int  number;
    char  name[20];
    char  sex;
    struct date birth;
    char  profession[6];
    char addr[30];
  };
```

13.1.3　结构体变量的使用

在编制 C 语言程序时，只能对结构体变量进行使用，即可对结构体变量赋值、存取或运算，而不能对一个结构体类型赋值、存取或运算。

1. 结构体变量的初始化

在定义结构体变量的同时给其赋初值，即为结构体变量的初始化。

例如：

```
struct teacher
  { int  number;
    char  name[20];
    char  sex;
    char  profession[6];
    char addr[30];
  };
  ……
  struct teacher t1 = {2001,"wang wei","T","docent","Xi'an"};
  struct teacher t2 = {2002,"lidong",'F',"profession","Xi'an"};
  ……
```

该例定义了两个结构体变量 t1 和 t2，并且让变量 t1 描述了“wang wei”教师，而变量 t2 描述了 li dong 教师。赋初值时应保证每个初值的类型与对应成员的类型一致，并且只能在定义结构体变量时使用该格式。

如下语句是错误的：

```
struct teacher  t1,t2;
t1 = {2001,"wang wei",'T',"docent","Xi'an"};
t2 = {2002,"lidong",'F',"profession","Xi'an"};
```

2. 结构体变量的使用

对于结构体变量的使用就像数组的使用，只能以分量的形式对结构体变量进行访问。如

同用下标法引用一个数组的元素。当然对结构体变量的成员引用不能通过下标法来实现，而要使用成员名进行访问。C 语言规定：可以对最低一级的成员进行引用。

对成员引用的一般格式：

结构体变量名 . 成员名

即引用结构体变量的各个成员时，要指出成员所属的结构体变量。“.”被称为分量运算符，用来获得一个结构体变量的某个成员。它是所有运算符中优先级最高的运算符。

[例 13-1] 利用分量运算符输出结构体各成员。

```
struct teacher
  { int   number;
    char  name[20];
    char  sex;
    char  profession[6];
    char  addr[30];
  };
main()
  { struct teacher t1,t2,t3;
    t1.number=2001;
    strcpy(t1.name,"wang wei");
    strcpy(t1.addr,"Xi'an");
    t.sex='T';
    strcpy(t,profession,"docent");
    printf("No:%ld,name:%s,sex:%c,profession:%s,addr:%s\n",t1.number,
t1.name,t1.sex,t1.profession,t1.addr);
    t2=t1;
    t3=t2;
  }
```

该程序的第 10～14 行和第 16 行引用的都是结构体变量 t1 的各个成员。第 10～14 行是给结构体变量 t1 的各个成员赋值。第 16 行是将 t1 变量值输出。

当两个结构体变量的类型相同时，故可以互相赋值。例如，例 13-1 中的第 17 行、第 18 行：

```
t2=t1;
t3=t2;
```

这两个语句是将变量 t1 的值赋给变量 t2，然后再将变量 t2 的值赋给变量 t3，从而使结构体变量 t1、t2 和 t3 具有相同的值。

对于嵌套型结构体变量的引用，是使用最内层的结构体成员参与运算（即只能对最低一级的成员进行引用）。使用时要注意成员的层次性，也是通过分量运算符“.”来一层一层获得最低一级成员引用的。

[例 13-2] 嵌套型结构体变量应用例子。

```
struct  date
  {int  year;
```

```
    int  month;
    int day;
  };
struct teacher
  { int  number;
    char name[20];
    char sec;
    struct date  birth;
    char  profession[6];
    char addr[30];
  };
main()
  {struct teacher t1,t2,t3;
    t1.number=2001;
    strcpy(t1.name,"wang wei");
    t1.sex='T';
    t1.birth.year=1970;
    t1.birth.month=10;
    t1.birth.day=23;
    strcpy(t1.profession,"docent");
    strcpy(t1.addr,"Xi'an");
    printf("No: %ld, name: %s, sex: %c, year: %d, month: %d, day: %d, profession: %s, addr: %s \
n",t1.number,t1.name,t1.sex,t1.birth.year,t1.birth.month,t1.birth.day,t1.profession,t1.addr);
  t2=t1;
  t3=t2;
  }
```

13.2　结构体数组

一个数组的各个元素都是一个结构体类型数据。这样的数组就是结构体数组。因而它与前面讲的数组概念一样，只不过该数组中每一个元素的类型是一个结构体类型。

13.2.1　结构体数组的定义

一般格式如下：

```
struct  结构体名  结构体数组名[大小];
```

其中，结构体名必须是已定义过的结构体类型，[大小] 是结构体数组元素的个数。

例如：

```
struct  teacher
  { int  number;
    char name[20];
```

```
        char sex;
        char  profession[16];
        char  addr[30];
      };
struct  teacher  a[20];
```

这里定义了一个结构体数组。它含有 20 个元素，每一个元素的类型都是 struct teacher 类型。

结构体数组也可以初始化，即在定义一个结构数组的同时给其赋值。

例如：

```
struct teacher
  { int number;
    char name[20];
    char sex;
    char profession[16];
    char addr[30];
    }t[3]={2001,"wang wei",'T',"docent","Xi'an",
           2002,"li dong",'T',"professor","xi'an",
           2003,"Zang tao",'T',"professor","xi'an"
          };
```

这里定义了一个结构体数组 t。它含有 3 个元素 t[0]，t[1] 和 t[2]，每一个元素的类型都是 struct teacher 类型，并且给 3 个元素赋了对应的值。

13.2.2 结构体数组的引用

结构体数组中的每一个元素相当于一个结构体变量，所以对结构体数组是通过其下标变量（元素）引用结构体成员的。其一般格式为：

结构体数组名的下标变量 . 成员名

[例 13-3] 通过结构体数组元素引用结构体成员。

```
struct teadcher
    {int number;
     char name[20];
     char sex;
     char profession[16];
     char addr[30];
    };
main()
{struct teacher t[3];
  for(i=0;i<3;i++)
  printf("\nNo:%d,name:%s,sex:%c,profession:%s,addr:%s\n",t[i].number,t[i].name,t[i]
.sex,t[i].profession,t[i].addr);
```

```
}
```

13.3　指向结构体类型数据的指针

一个指向结构体变量的指针就是该结构体变量所占据的内存段的起始地址。而结构体变量名就是指向该内存段起始地址的指针。它是一个指针常量。也可以设一个指针变量，用来指向一个结构体变量，这样就可以用该指针变量引用结构体变量成员。

13.3.1　指向结构体变量的指针定义

结构体变量指针的定义格式为：

```
结构体类型名　　* 指针变量名;
```

[例 13-4]　指向结构体类型指针变量的应用。

```
struct teacher
  { int number;
  char name[20];
  char sex;
  char prifession[16];
  char addr[30];
};
main()
{ struct teacher t1 = {2001,"wang wei","T","docent","Xi'an"};
  struct teacher t2 = {2002,"lidong",'F',"profession","Xi'an"};
  struct teacher * pt1;
  pt1 = &t2;
  ......
}
```

在例 13-4 中，定义了一个指针变量 pt1，并让它指向了结构体变量 t2。注意：上面程序段里语句“pt1 = &t2;”中的 &t2 不能写成 t2。因为 t2 是结构体变量，故应使用取地址运算符 & 来获得该变量的地址值。

当一个结构体指针定义好后，它就可以被赋值而指向一个结构体变量或结构体数组，对结构体指针的运算同一般指针的运算，如加 1 或减 1。但加（或减 1）是使指针指向下一个（或上一个）结构体变量（如果存在）。

[例 13-5]　指向结构体类型指针变量的应用 2。

```
struct date
  { int year;
    int month;
    int dat;
  };
struc point
```

```
    int x;
      int y;
    };
main()
    {struct date d1,d2, * pd;
      struct point b1,b2[20], * pb;
      pd = &d2;
      pb = b2
      ……
    }
```

例 13-5 中定义了两个指针变量 pd 和 pb，同时给这两个指针变量也赋了值（pd 指向结构体变量 d2，pb 指向结构体数组 b2）。如果在程序中有“pd++;”语句，则指针指向了结构体变量 d2 所占存储空间之后，如图 13-1（a）所示。而如果在程序中有“pb++;”语句，则指针 pb 指向了结构体数组 b2 的 b2［1］元素，如图 13-1（b）所示。

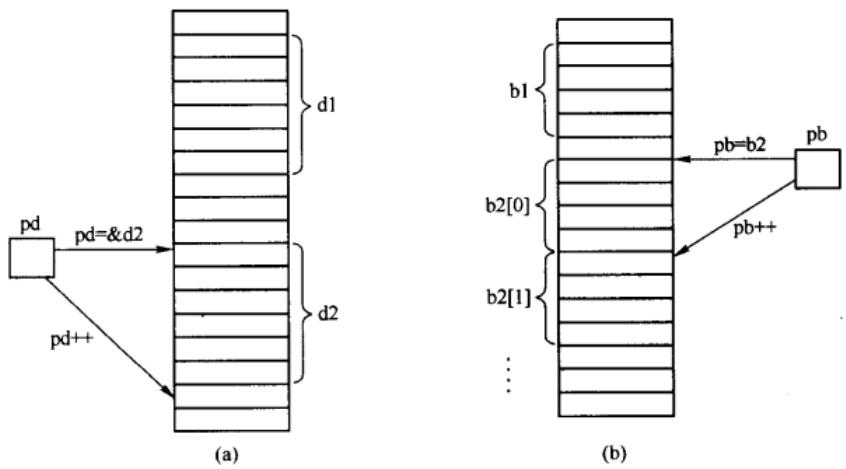


图 13-1 结构体指针变量运算示意图

从结构体指针变量示意图可以看出一个结构体指针所指向的结构体变量，必须与它定义时所规定的结构体类型一致，不能指向与它的类型不一致的其他结构体变量；否则，在指针运算时将会出错。

13.3.2 用指针引用结构体成员

前面已介绍了引用结构体变量的各个成员时，可用“.”运算符来获得一个结构体变量某个成员的方法。当一个结构体指针变量指向一个结构体变量时，就可以用该指针引用结构体的成员。其引用格式为：

（*结构体指针名）.成员名

注意：该引用格式中的圆括号在使用时不能省略。

[例 13-6] 用结构体指针建立一个通讯录。

```
#include  "stdio.h"
#define  M  50
struct  tongxin
{char  name[20];
  char sex;
  int age;
  char  tel[8];
  char  addr[80];
};
main()
{struct tongxin  man[M], *p;
  int  i;
  p=man;
  for(i=0;i<M;i++)
    {printf("\nEnter  Name:");gets((*p).name);
      printf("\nEnter  age:");scanfc"%d",&(*p.age);
      printf("\n Enter sex:";getc((*p).sex);
      printf("\nEnter Tel:");gets((*p).tel);
      printf("\nEnter addr:");gets((*p).addr);
      p++;
  }
p=man;
for(i=0;i<M;i++)
  printf("\n%s,%c,%d,%s,%s",(*p).name (*p).sex,(*p).age,(*p).tel,(*p).addr);
p++;
}}
```

在用指针引用结构体成员时，*p 两侧的括号不能省略，因为成员运算符“.”优先于“*”运算符。

C语言中还可以通过“->”引用成员，如：

```
p->name 等价于(*p).name
p->sex 等价于(*p).sex
p->age 等价于(*p).age
```

再看一种结构体成员为指针的情况，如：

```
struct lpoint
{ int x;
  int *y;
}*q,s[3];
```

这里定义了一个结构类型 lpoint。它含有两个成员：一个 int 型变量 x；一个 int 型的指针变量 y。同时还定义了一个具有 struct lpoint 类型的结构体指针变量 q。因而：

*q->y	表示 y 所指的内容；
*q->y++	访问 y 所指的内容后 y 自增
(*q->y)++	自增 y 所指的内容；
*q++ ->y	访问 y 所指的内容后 q 自增

注意：“->”的优先级与“()”、“[]”和句点“.”相同。优称于“*”运算符。

13.4 共用体

13.4.1 共用体类型的定义

共用体类型是一种用户自定义的构造类型，和结构体一样，是由不同类型的、固定个数的成员组成的。但结构体的成员是同时存在的，并分别占用不同的内存单元。而对于共用体的所有成员而言，所有的共用体成员共占同一段内存。从微观上看，某一时刻只能有某一成员起作用，其他成员不存在。

共用体类型用来表示在不同场合会有不同类型的值的对象。若按照它的各种取值的类型去分散定义多个变量来表示这个对象：一方面会浪费存储空间，因为它在一瞬时只有一种固定类型的取值，还会同时有多个值；另一方面，用多个变量来表示一个对象会人为地割裂对象的整体性。

共用体类型定义的形式为：

```
union  共用体名
{ 类型名 1  成员名;
  类型名 2  成员名;
  ……
  类型名 n    成员名 n;
};
```

说明：

union 是关键字，标志共用体类型，union 后面跟的是共用体类型的名字，共用体名应符合标识符的命名规则；括号后是共用体的各个成员，定义成员的方式和定义变量相同，各成员之间用分号分隔，共用体类型的定义以分号结束；共用体类型的成员可以是基本类型的，也可以是构造类型的，但在通常情况下，共用体的成员是基本类型的；共用体的所有成员共占一段内存，所有成员的起始地址是相同的，整个共用体所占的内存单元的大小等于需要占用内存单元最多的那个成员所占的单元数，如：

```
union  un
  {char  c;
  int k;
  float f;
};
```

上述定义表示共用体类型 un 有 3 个成员。其中，c 是字符型的，需占 1 个字节；k 是整型的，需占 2 个字节；f 是单精度实型的，需占 4 个字节。那么，这种共用体类型需要 4 个字节，并且这 3 个成员所占单元的起始地址和共用体的起始地址都是相同的。也就是说，若

c 存在占用第 1 个字节，则 k 存在占用第 1 个和第 2 个字节，f 存在占用所有的 4 个字节。

13.4.2　共用体变量的定义

和结构体一样，定义了共用体类型只表示定义了一种类型。变量才是存放数据的实体。定义和引用共用体变量的方式和结构体变量的定义、引用类似，也有 3 种。

(1) 用已定义的共用体类型去定义共用体变量

```
union  un
  {char  c;
    int  k;
    float f;
};
union  un  un1,un2;
```

(2) 定义共用体类型的同时定义变量

```
union  un
  {char c;
    int k;
    float f;
}un1,un2;
```

(3) 定义无名共用体的类型的同时定义变量

```
union
  {char  c;
    int k;
    float f;
} un1,un2;
```

指向共用体的指针变量的定义，与指向结构体的指针变量的定义方式是相同的，在此不再赘述。

13.4.3　共用体变量的引用

对于共用体变量只能引用它的成员，而不能引用整个共用体变量。共用体变量成员的引用方式为：

```
共用体变量名 . 成员名
```

例如：

```
un1.c,un1.f,un1.k
```

若共用体的成员是构造类型的，则需要逐级引用至最低级的成员。

同样，也能通过定义指向共用体的指针变量来引用共用体变量。如已有定义：

```
union  un  * pu= &un1;
```

那么，以下是合法的引用方式，即：

```
(*pu).c,pu->k
```

共用体变量的地址与共用体成员的地址是相同的，如对上面定义的共用体变量 un1 而言，&un1，&un1.c，&un1.k，&un1.f 是相等的，但是它们的类型是不同的。

在给共用体变量赋值时，只对共用体的一个成员赋值，而不能对整个变量赋值，也不能对共用体变量在定义时初始化。因为共用体的成员共用一段连续的内存单元，若多次给共用体的成员赋值，那么每次赋值都会覆盖原来的值。也就是说，对共用体变量而言，只有最近一次的赋值是有效的，而再去使用原来成员的值是不合适的。

[例 13-7] 分析下列程序的输出结果，理解共用体类型变量的使用特点。

```
union  uu
  {int a[3][2];        /*第1个成员a为整型的二维数组,占用12个字节*/
    char  c[10];       /*第2个成员c为字符型一维数组,占用10个字节*/
    double d;          /*第3个成员d为双精度实型,占用8个字节*/
  }
main()
{union uu u1;
  int i,j,k;
  u1.c[0]='A';u1.c[1]='B';u1.c[2]='C';u1.c[3]='\0';      /*对第2个成员(字符数组)
赋值*/
  printf("\nthe string is :%s",u1.c);          /*输出字符数组存放的字符串*/
  k=0;
  for(i=0;i<3;i++)
    {k++;
    for(j=0;j<2;j++)
      u1.a[i][j]=k;
  }                                            /*给第1个成员(二维数组a)按行赋值*/
printf("\nthe array id :");
for(i=0;i<3;i++)
  {printf("\n");
    for(j=0;j<2;j++)
      printf("%3d",u1.a[i][j]);
}                                              /*按行输出第1个成员(二维数组a)的元素*/
  u1.d=3.1415926;                              /*给第3个成员d赋值*/
  printf("\nthe real is %8.4lf",u1.d);         /*输出第3个成员的值*/
}
```

本程序中定义的共用体变量有 3 个成员，共占 12 个字节的内存空间，在某一时刻只有一个成员是有效的。在程序中，首先引用第 2 个成员——字符数组 c，向元素逐个赋值，然后输出其中存放的字符串；接下来，引用第 1 个成员——二维数组 a，采用 for 循环给元素赋值，这时字符数组 c 就失效了，然后按行输出各个元素；最后给第 3 个成员——双精度实型变量 d 赋值，这时只有 d 是有效的，前面给其他成员赋的值已被覆盖，不再存在。

13.5　枚举类型

13.5.1　枚举类型的定义

在现实问题中，有些数据的取值用有意义的名字表示更直观，如月份、星期用相应的单词来表示比用整型或字符型表示能使得程序的可读性更好，并且取值应该限定在合理的范围内。

在C语言中，枚举类型就是若干个采用标识符表示的整型常量的集合。枚举类型的定义方式为

```
enum　枚举类型名{枚举常量1,枚举常量2,……,枚举常量n};
```

说明：

enum是用来定义枚举类型的关键字。枚举类型名是用户定义的类型名。枚举常量是用户定义的有意义的标识符，即罗列出该枚举类型各种可能的、合法的取值。

例如定义表示月份的类型：

```
enum　months{jan,feb,mar,apr,may,jun,jul,aug,sep,oct,nov,dec};
```

例如定义表示星期的类型：

```
enom week{sun,mon,tue,wed,thu,fri,sat};
```

枚举常量具有确定的值，每一个枚举常量隐含着一个整型的值；在默认的情况下，第1个枚举常量的值是0，以后每一个枚举常量的值是前一个常量的值加1。如上面的月份的定义，各枚举常量的值从0～11。

也可以人为地指定枚举常量的值，如

```
enum　months{jan=1,feb,mar,apr,may,jun,jul,aug,sep,oct,nov,dec};
```

枚举常量被显示地赋值为1，后面的常量的值和前一个相比逐个增1，即在本类型中，各枚举常量的值为1～12。再如：

```
enom　week{sun=7,mon=1,tue,wed,thu,fri,sat};
```

sun被指定为7，mon被指定为1，后面的常量tue的值为2，wed的值为3，thu的值为4，fri的值为5，sat的值为6。

定义时，显式指定枚举常量的值，其实是指定枚举常量间的顺序。在定义枚举类型后，对于已定义的枚举常量是不能再赋值的，因为常量的值是确定的，不能被修改。

13.5.2　枚举类型变量的定义和使用

1. 枚举类型变量的定义

枚举类型变量的定义也有3种方式：

↘ 用已定义的枚举类型去定义变量：

```
enum　week　w1,we[3];
```

↘ 在定义枚举类型的同时定义变量：

```
enum  week{sun,mon,tue,wed,thu,fri,sat}w1,we[3];
```

↘ 对于第 2 方式也可省略枚举类型名：

```
enum  {sun,mon,tue,wed,thu.fri,sat}w1,we[3];
```

2. 枚举类型变量的引用

枚举类型变量的引用方式和普通变量一样，但枚举类型变量的取值范围只能是其枚举类型所枚举的各个常量，一般给枚举类型变量赋值为枚举常量。

虽然编译系统将枚举类型处理为整型，但整型和枚举类型之间不能相互赋值；若要赋值，需进行强制类型转换，如：

```
w1 = mon;
we[0] = (enoum week)1;
```

都是合法的赋值方式，而“we [2] =2;”则不是合法的赋值方式。

可以比较枚举类型变量的大小，枚举变量的比较相当于比较它们所隐含的整数值，如已有赋值：

```
we[0] = mon;we[1] = sat;那么 we[0]<we[1]成立;
```

枚举类型变量的值是在该枚举类型中枚举的常量所代表的整数值范围内的，因此相应变量可以作为循环变量来控制循环。

[例 13-8] 输出 12 个月份的英文单词。

```
enum  months{jan=1,feb,mar,apr,may,jun,jul,aug,sep,oct,nov,dec};
main()
{ enum  months  m;
char
 * name[ ] = {"Jannuary","February","March","April","May","June","July","August","September",
"October","November","December"};
for(m=jan;m<=dec;m++)
  printf("\n%2d;%s",m,name[m-1]);
}
```

13.6 用户定义类型

13.6.1 用户自定义类型的含义和格式

在 C 语言中，允许用户自己用新的类型名去声明已有的类型。用户自定义类型是对已经存在的标准类型或已经定义了的类型起一个新名字，而不是去定义原来没有的类型，因此用户自定义类型实际上是一种对类型的声明。

声明用户自定义类型的变量后，将变量名用新类型名替换，最后在最前面加上 typedef。这样一个新类型就声明了，并可以用其去定义变量。

13.6.2　用 typedef 声明基本类型

声明基本类型的形式为：

```
typedef　基本类型名　新类型名;
```

例如：

typedef int　INTEGER 表示用类型名 INTERGER 来代表类型名 int；

typedef　char CHARACTER 表示用类型名 CHARACTER 来代表类型名 char。

有了上述定义后，就可以用新声明的类型名去定义变量，如：

INTEGER a，b；表示定义整型变量，它与定义语句 int a，b；等价；

CHARACTER c1，c2；表示定义字符型变量，它与定义语句 char c1，c2；等价。

13.6.3　用 typedef 声明构造类型

1. 数组类型的声明

声明数组类型的类型名的形式为：

```
数据类型名　　数组类型名[数组长度];
```

例如：

```
typedef　int　ARRAY[10];　/* 声明是一个长度为 10 的整型数组类型 */
```

可以用去定义数组类型的变量。例如：

```
ARRAY a,b;
```

那么，a、b 就是长度为 10 的整型数组变量，相当于：

```
int a[10],b[10];
```

2. 结构体类型的声明

结构体类型的声明形式为：

```
typedef　struct
  {类型名　成员名 1;
   类型名　成员名 2;
   ……
   类型名　成员名 n;
  }结构体类型名;
```

例如：

```
typedef struct
  {int　year;
   int month;
   int day;
  }DATE;                          /* 声明 DATE 为结构体类型名 */
  DATE　today,yesterday;          /* 定义 today,yesterday 为 DATE 类型变量 */
```

上述定义等价于：

```
struct
  {int  year;
   int  month;
   int  day;
  }today,yesterday;
```

3. 共用体类型的声明

共用体类型声明的形式为：

```
typedef  union
  {类型名  成员名 1;
   类型名  成员名 2;
   ……
   类型名  成员名 n;
  }共用体类型名;
```

例如：

```
typedef  union
  {int  k;
   char  c;
   float  f;
  }DA;                    /* 声明 DA 为共用体类型名 */
  DA  d1,d2,d3;           /* 定义 d1,d2,d3 为 DA 类型变量 */
```

小结

在 C 语言中，结构体、共用体和枚举类型是用户自定义类型，利用这三种类型数据可以处理复杂的问题。

本章主要由结构体、共用体、枚举类型及用户自定义类型四部分组成。

在结构体部分中，主要介绍的是结构体变量的定义、引用和初始化，结构体数组的定义和引用。需重点掌握的是定义结构体类型、结构体成员运算符及结构体数组。

结构体是把具有相互关系的不同类型的数据组成一个有机整体。当说明一个结构体变量时，系统分配给它的内存是成员所需内存量的总和，对结构体变量的引用必须是对其成员的引用 。

在共用体部分，主要介绍的是定义共用体类型、共用体变量的定义和引用。需要重点掌握的是共用体变量的引用。

共用体又被称为联合体，使几种不同类型的变量共用一段存储空间。当说明一个共用体变量时，系统分配给它的内存是各成员中占内存量最大者所需的容量。对共用体变量的引用也必须是对其成员的引用。共用体变量在程序运算期间只有一个成员驻留在内存中。

在枚举部分中，主要介绍的是定义枚举类型和枚举变量的使用。

枚举类型将变量的值一一列举出来。枚举类型中的元素是常量。

在用户自定义类型部分中，主要介绍的是利用 typedef 来定义的新数据类型。所定义的新数据类型只是代替已有的类型名。

习题

1. 定义一个结构体变量（包括年、月、日）。计算该日在本年中是第几天？注意闰年问题。

2. 编写程序，输入 7 个职工的号码、姓名、性别、基本工资、奖金、水电费，计算每个人的实发额（基本工资 + 奖金 - 水电费），输出实发额最高和最低的职工的姓名和实发额。

3. 某单位对职工进行专业考核，50 岁以下的职工进行闭卷考试，成绩以百分制记录，60 分以下为不及格；50 岁以上的职工进行开卷考试，成绩分为 A，B，C 3 种，C 为不及格。每个人的成绩只有一种形式的记录。编写程序，输出所有及格职工的考号、姓名、成绩及所有不及格职工的考号、姓名。

4. 输入一个人的生日和当前日期，计算实足年龄。日期采用结构体类型变量存放。

第 14 章　文　　件

本章要点

- ☑ 文件的基本概念
- ☑ 文件类型指针
- ☑ 文件的打开与关闭
- ☑ 文件的读写
- ☑ 文件的定位

14.1　C 文件概述

文件是程序设计中的一个重要概念。所谓“文件”，一般是指存储在外部介质上数据的集合。一批数据是以文件的形式存放在外部介质（如磁盘）上的。操作系统是以文件为单位对数据进行管理的。也就是说，如果想找存储在外部介质上的数据，则必须先按文件名找到所指定的文件，然后再从该文件中读取数据。要向外部介质上存储数据也必须先建立一个文件（以文件名标识），才能向它输出数据。

从操作系统的角度来看，每一个与主机相联的输入设备、输出设备也都是一个文件，称其为设备文件或外设文件。例如，键盘是一个输入文件，显示器是一个输出文件。

程序运行常常需要将一个数据（运行的最终结果或中间数据）输出到磁盘上存放起来，以后需要时再从磁盘中输入到计算机内存。这就要用到磁盘文件。

C 语言把文件看作是一个字符（字节）的序列，即由一个一个字符（字节）的数据顺序组成。根据数据的组织形式，可分为 ASCII 文件和二进制文件。ASCII 文件又被称为文本文件。它的每一字节均放一个 ASCII 代码，代表一个字符。二进制文件是把内存中的数据按其在内存中的存储形式原样输出到磁盘上存放。如果有一个整数 10000，在内存中占两个字节，按 ASCII 码形式输出，则占 5 个字节，而按二进制形式输出，在磁盘上只占两个字节，如图 14-1 所示。用 ASCII 码形式输出与字符一一对应，一个字节代表一个字符，因而便于对字符进行逐个处理，也便于输出字符。但一般占存储空间较多，而且花费转换时间（二进制形式与 ASCII 码间的转换）。用二进制形式输出数值，可以节省外存和转换时间，但一个字节并不对应一个字符，不能直接输出字符形式。一般中间结果数据需要暂时保存在外存上以后又需要输入到内存的，常用二进制文件保存。

因此，一个 C 文件是一个字节流或二进制流。它把数据看作是一连串的字符（字节），而不考虑记录的界限。换句话说，C 语言中文件并不是由记录组成的。在 C 语言中对文件的存取是以字符为单位的。输入输出数据流的开始和结束仅受程序控制而不受物理符号控制

(如换行回车符)。也就是说，在输出时不会自动增加回车换行符以作为记录结束的标志，输入时也不以回车换行符作为记录的间隔（实际上，C 文件并不由记录构成)。我们把这种文件称为流式文件。C 语言允许对文件存/取一个字符，这就增加了处理的灵活性。

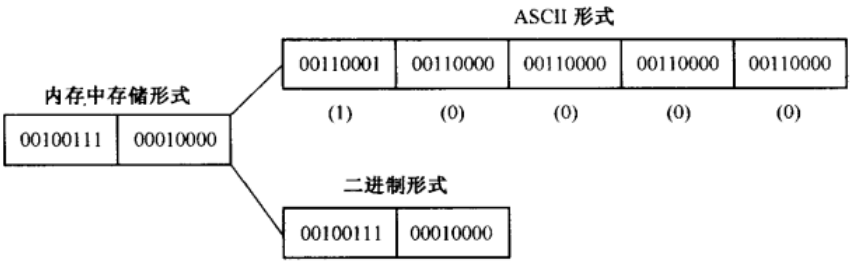


图 14-1　两种文件存储方式

不同的 C 语言版本有两种对文件的处理方法：一种叫“缓冲文件系统”；一种叫“非缓冲文件系统”。所谓缓冲文件系统是指系统自动地在内存区为每一个正在使用的文件开辟一个缓冲区。从内存向磁盘输出数据必须先送到内存中的缓冲区，装满缓冲区后才一起送到磁盘中。如果从磁盘向内存读入数据，则一次从磁盘文件将一批数据输入到内存缓冲区（充满缓冲区)，然后再从缓冲区逐个地将数据送到程序数据区（给程序变量)，如图 14-2 所示。缓冲区的大小由各个具体的 C 版本确定，一般为 512 字节。

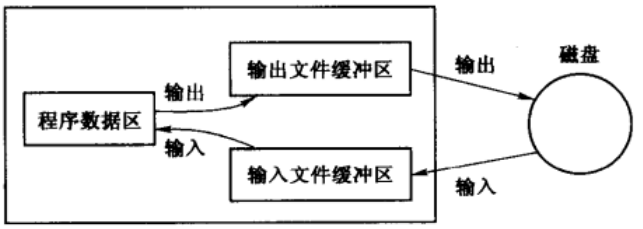


图 14-2　缓冲文件存储过程

所谓“非缓冲文件系统”是指系统不自动开辟确定大小的缓冲区，而由程序为每个文件设定缓冲区。

一般地，用缓冲文件系统进行的输入输出被称为高级磁盘输入输出；用非缓冲文件系统进行的输入输出被称为低级（低层）的输入输出。ANSI C 标准决定不采用非缓冲文件系统，而只采用缓冲文件系统。

在 C 语言中，对文件的操作都是通过标准函数来实现的。文件的打开、读写、关闭等都是这样。也就是说，学习 C 语言的文件操作，主要是掌握有关文件的操作函数。

14.2　文件类型指针

在缓冲文件系统中，关键的概念是“文件指针”。每个被使用的文件都在内存中开辟一个缓冲区来存放文件的有关信息（如文件的名字、文件状态及文件当前位置等)。这些信息是保存在一个结构体变量中的。该结构体类型是由系统定义的，取名为 FILE。Turbo C 在 stdio.h 文件中有以下的文件类型声明：

```
typrdef  struct
{ short  lever;                    /* 缓冲区"满"或"空"的程序 */
  unsigned  flags;                 /* 文件状态标志 */
  char  fd;                        /* 文件描述符 */
  unsigned  char hold;             /* 如无缓冲区则不读取字符 */
  short  bsize;                    /* 缓冲区的大小 */
  unsigned char  * buffer;         /* 数据缓冲区的位置 */
  unsigned ar curp;                /* 指针,当前的指针 */
  unsigned istemp;                 /* 临时文件,指示器 */
  short  token;                    /* 用于有效性检查 */
}FILE;
```

有了结构体 FILE 类型之后，就可以用它来定义若干个 FILE 类型的变量，以便存放若干个文件的信息。一般地，可以定义文件指针变量，用以指向 FILE 类型的变量。例如：

```
FILE  * fp;
```

fp 就是一个指向 FILE 类型结构的指针变量，通过 fp 使用结构体变量中的文件信息，从而访问文件。也就是说，C 程序中是通过文件指针变量找到与它相关的文件的。

如果有 n 个文件，则一般应设 n 个指针变量（指向 FILE 类型的结构体的指针变量），以实现对该文件的访问。

14.3 文件的打开与关闭

与其他高级语言一样，对文件读写之前应该先“打开”该文件，在使用结束之后应“关闭”该文件。

14.3.1 文件的打开（fopen 函数）

ANSI C 规定了标准的输入输出函数库，用 fopen（）函数来实现打开文件。fopen（）函数的调用方式通常为：

```
FILE * fp;
fp=fopen(文件名,使用文件方式);
```

例如：

```
fp=fopen("a1","r");
```

它表示要打开名字为 a1 的文件，使用文件方式为“读入”（r 代表 read，即读入）。fopen 函数带回指向 a1 文件的指针并赋给 fp，这样 fp 就和文件 a1 相联系了。或者说，fp 指向 a1 文件。可以看出，在打开一个文件时，通知编译系统 3 个信息：

- 需要打开的文件名，也就是准备访问的文件的名字；
- 使用文件的方式（读还是写等）；
- 让哪一个指针变量指向被打开的文件。

文件的使用方式及其含义如下。

“r”为读打开一个文本文件。在此方式打开的文件只能用于向计算机输入而不能用作向该文件输出数据，而且该文件应该已经存在。若指定的文件不存在，则会出错。

“w”为写打开一个文本文件。以写方式打开的文件只能用于向该文件写数据（即输出文件），而不能用来向计算机输入。如果原来不存在该文件，则在打开时新建立一个以指定的名字命名的文件。如果原来已存在一个以该文件名命名的文件，则在打开时将该文件删除，然后重新建立一个新文件。

“a”为向文本文件尾追加数据。此时，该文件必须已存在；否则，将得到出错信息。打开时，位置指针移到文件末尾。

“r+”，“w+”，“a+”表示用“r+”，“w+”，“a+”方式打开文件，既可以用来输入数据，也可以用来输出数据。用“r+”方式时，该文件应该已经存在，以便能向计算机输入数据。用“w+”方式则新建立一个文件，先向此文件写数据，然后可以读此文件中的数据。用“a+”方式打开的文件，原来的文件不被删除，位置指针移到文件末尾，可以添加，也可以读。

“rb”，“wb”，“ab”表示以读、写、追加的方式打开一个二进制文件，使用方法同“r+”，“w+”，“a+”。

“rb+”，“wb+”，“ab+”表示以读、写、追加的方式打开一个二进制文件，使用方法同“r+”，“w+”，“a+”。

如果不能实现“打开”的任务，则 fopen 函数将会带回一个出错信息。出错的原因可能是：用“r”方式打开一个并不存在的文件；磁盘出故障；磁盘已满无法建立新文件；等等。此时，fopen 函数将带一个空指针 NULL（NULL 在 stdio.h 文件中已被定义为 0）。

常用下面的方法打开一个文件：

```
if((fp=fopen("file1","r"))==NULL)
  {printf("cannot open this files\n");
    exit(0);
  }
```

即先检查打开的操作是否出错，如果有错，就在终端上输出“cannot open this file”。exit 函数的作用是关闭所有文件，终止正在调用的过程。待用户检查出错误并修改后再运行。

用以上的方式可以打开文本文件或二进制文件，这是 ANSI C 的规定，用同一种缓冲文件系统来处理文本文件和二进制文件。但目前使用的有些 C 编译系统可能不完全提供所有这些功能（如有的只能用“r”，“w”，“a”方式），有的 C 版本不用“r+”，“w+”，“a+”，而用“rw”，“wr”，“ar”等，注意所用系统的规定。

在向计算机输入文本文件时，将回车换行符转换为一个换行符，在输出时，把换行符转换成为回车和换行两个字符。在用二进制文件时，不进行这种转换，在内存中的数据形式与输出到外部文件中的数据形式完全一致，一一对应。

在程序开始运行时，系统自动打开 3 个标准文件，即标准输入、标准输出及标准出错输出。通常，这 3 个文件都与终端相联系。因此，以前我们所用到的从终端输入或输出都不需要打开终端文件。

14.3.2 文件的关闭（fclose 函数）

在使用完一个文件后应该关闭它，以防止它再被误用。“关闭”就是使文件指针变量不指向该文件，也就是文件指针变量与文件“脱钩”，此后不能再通过该指针对原来与其相联系的文件进行读写操作。除非再次打开，使该指针变量重新指向该文件。

用 fcolse 函数关闭文件。fcolse 函数调用的一般形式为：

```
fcolse（文件指针）
```

例如：

```
fclose(fp);
```

前面曾把打开文件（用 fopen 函数）时所带回的指针赋给了 fp，现今通过 fp 把该文件关闭，即 fp 不再指向该文件。

应该养成在程序终止之前关闭所有文件的习惯。如果不关闭文件，则将会丢失数据。因为，如前所述，在向文件写数据时，是先将数据输出到缓冲区，待缓冲区充满后才正式输出给文件。如果当数据不充满缓冲区而程序结束运行时，就会将缓冲区中的数据丢失。用 fclose 函数关闭文件，可以避免这个问题。它先把缓冲区中的数据输出到磁盘文件，然后才释放文件指针变量。

fcolse 函数也带回一个值，当顺利地执行了关闭操作后，则返回值为 0；否则，EOF（-1），可以用函数来测试。

14.4 文件的读写

打开文件后，就可以对它进行读写了。常用的读写函数包括字符读写、数据读写、格式化读写、字读写及字符读写等方式。

14.4.1 fputc 函数和 fgetc 函数

1. fputc 函数

fputc 函数的作用是把一个字符写到磁盘文件上去。其一般调用形式为：

```
fputc(ch,fp);
```

其中，ch 是要输出的字符，可以是一个字符常量，也可以是一个字符变量；fp 是文件指针变量。fputc（ch，fp）函数的作用是将字符 ch 输出到 fp 所指向的文件中去。fputc 函数也带回一个值。如果输出成功，则返回值就是输出的字符。如果输出失败，则返回一个 EOF（-1）。EOF 是在 stdio.h 文件中定义的符号常量，值为 -1。

其实，putchar 函数是从 fputc 函数派生出来的，是在 stdio.h 文件中用预处理命令定义的宏：

```
#define putchar fputc(ch,stdout)
```

前面已叙述，stdout 是系统定义的文件指针变量。它与终端输出相连。fputc（ch，

stdout)的作用是将 ch 的值输出到终端。用宏写简单一些。从用户的角度，可以把它看为函数而不必严格地称其为宏。

2. fgetc 函数

从指定的文件读入一个字符。该文件必须是以读或读写方式打开的。fgetc 函数的调用形式为：

```
ch=fgetc(fp);
```

其中，fp 为文件类型指针变量；ch 为字符变量。fgetc 函数带回一个字符，赋给 ch。如果在执行 fgetc 函数读字符时遇到文件结束符，则函数返回一个文件结束标志 EOF，即（-1）。如果想从一个磁盘文件顺序读入字符并在屏幕上显示出来，则可以使用下述程序，即：

```
ch=fgetc(fp);
while(ch! =EOF)
    {putchar(ch);
    ch=fgetc(fp);
  }
```

注意：EOF 不是可输出字符，因此不能在屏幕上显示。

如果想顺序读入一个二进制文件中的数据，则可以用如下程序：

```
while(! feof(fp))
  {ch=fgetc(fp);
    ……
  }
```

如不遇到文件结束符，则 feof（fp）的值为 0，! feof（fp）的值为 1，读入一个字节的数据赋给整型变量，并接着对其进行所需的处理。直到遇到文件结束，feof（fp）的值为 1，! feof（fp）值为 0，不再执行循环。

3. fputc 函数和 fgetc 函数的应用举例

[例 14-1]　从键盘输入一些字符，逐个把它们送到磁盘上去，直到输入一个“#”为止。

```
#include <stdio.h>
main()
{ FILE *fp;
  char ch ,filename[10];
  scan("%s",filename);
  if((fp=fopen(filename,"w"))==NULL)
    {printf("cannot open file\n");
    exit(0);
    }
  ch=getchar();
  ch=getchar();
  while(ch! ='#')
    {fputc(ch,fp);putchar(ch);
```

```
        ch=getchar();
      }
    fclose(fp);
}
```

其运行情况为：

```
file1.c                (输入文件名)
computer  and  c#(输入一个字符串)
computer  and  c(输入 一个字符串)
```

[例 14-2] 将一个磁盘文件中的信息复制到另一个磁盘文件中。

```
#include  <stdio.h>
main()
{FILE  *in, *out;
  char  ch,infile[10],outfile[10];
  printf("Enter  the  infile name: \n");
  scanf("%s",infile);
  printf("Enter the outfile name: \n");
  scanf("%s",outfile);
  if((in=fopen(infile,"r"))==NULL)
    {printf("cannot  open  infile\n");
      exit(0);
    }
  if((out=fopen(outfile,"w"))==NULL)
    {printf("cannot  open  outfile\n");
      exit(0);
    }
  while(! feof(in))  fputc(fgetc(in),out);
  fclose(in);
  fclose(out);
}
```

其运行情况为：

```
Enter the infile name:
file1.c                (输入原有磁盘文件名)
Enter the outfile name:
file2.c                (输入新复制的磁盘文件名)
```

14.4.2 fread () 函数和 fwrite () 函数

用 getc 和 putc 函数可以用来读写文件中的一个字符，但是常常要求一次读入一组数据(如一个实数或一个结构体变量的值)。ANSI C 标准提出设置两个函数，用来读/写一个数据块。

1. fread 函数

fread 函数的一般形式为：

```
fread(buffer,size,count,fp);
```

其中，buffer 是一个指针，是读入数据的存放地址，输入的数据存入该内存块；size 为要读写的字节数；count 是要进行读写多少字节的数据项；fp 为文件型指针。

fread 函数的功能为：从 fp 指向的文件中读取 size 个 count 个字节大小的数据块，存放到 buffer 指定的内存中。如果 fread 调用成功，则函数返回值为 count 的值，即读入数据项的个数。如果遇到文件结束符或出错，则返回 0。

2. fwrite 函数

fwrite 函数的一般形式为：

```
fwrite(buffer,size,count,fp);
```

其中，buffer 是准备输出的数据块的首地址；size 和 count 的含义同 fread 函数。

fwrite 函数的功能为：将 buffer 指向的内存中的 count×size 个字节内容写入到 fp 指定的文件中。函数返回值为 count 的值，即输入到文件的数据项个数。

说明：如果文件以二进制形式打开，则用 fread 和 fwrite 函数就可以读/写任何类型的信息。

3. fread 函数和 fwrite 函数的使用举例

[例 14-3]　假设一个学生的信息包括姓名、学号、年龄及成绩，写一程序完成从键盘读入 40 名学生的信息，然后把信息转存到磁盘文件中去。

根据要求，学生的信息应该是结构体类型，40 名学生信息可以由一个结构体数组表示，即：

```
struct student_type
{char  name[10];
  int  num;
  int age;
  float mark;
} stude[40];
```

使用以下 for 语句和 fwrite 函数可以将内存中的学生数据输出到磁盘文件中去，即：

```
for(i=0;i<40;i++)
  fwrite(&stud[i],sizof(struct student_type),1,fp);
```

其完整的程序如下：

```
#include  <stdio.h>
#define  SIZE  40
struct student_type
{char  name[10];
  int  num;
  int age;
```

```
  float  mark;
| stude[40];
viod  save()
{FILE  * fp;
  int  i;
if((fp=fopen("stu_list","wb"))==NULL)
  {printf("cannot  open  file\n");
    return;
  }
for(i=0;i<SIZE;i++)
  if (fwrite(&stude[i],sizeof(struct student_type),1,fp)!  =1)
    printf("file  write  error\n");
  fclose(fp);
}
main()
{int i;
  for(i=0;i<SIZE;i++)
  scanf("%s%d%d%f",stud[i].name,&stud[i].num,&stud[i].age,&stud[i].mark);
  save();
}
```

在 main 函数中，从终端键盘输入 40 名学生的数据，然后调用 save 函数，将这些数据输出到以“stu_list”命名的磁盘文件中。fwrite 函数的作用是将一个数据块送到文件中。

14.4.3 fprintf 函数和 fscanf 函数

fprintf 函数和 fscanf 函数是文件的格式化读写函数。

fprintf 函数、fscanf 函数与 printf 函数、scanf 函数的作用相仿，只是 fprintf 函数和 fscanf函数的读写对象不是终端而是磁盘文件。它们的一般调用方式为：

```
fscanf(文件指针,格式字符串,输入表列);
fprintf(文件指针,格式字符串,输出表列);
```

例如：

```
fprintf(fp,"%d,%6.2f",i,t);
```

其作用是将整形变量 i 和实型变量 t 的值按%d 和%6.2f 的格式输出到指向的文件中。如果 i=3，t=4.5,则输出到磁盘文件中的是以下字符串，即：

```
3,4.50
```

再如：

```
fscanf(fp,"%d,%f",&i,&t);
```

它的作用是从 fp 指向的文件中按%d 和%f 格式读入数据。注意文件中存放的是 ASCII 码字

符，在读入过程中自动转换成格式要求的数据。

磁盘文件上如果有以下字符：

```
3,4.5
```

则将磁盘文件中的字符“3”转换成整数 3 送给变量 i，将字符“4.5”转换成实数 4.5 送给变量 t。

用 fpritf 函数和 fscanf 函数对磁盘文件读写，使用方便，容易理解，但由于在输入时要将 ASCII 码转换为二进制形式，在输出时又要将二进制形式转换成字符，花费时间比较长。因此，在内存与磁盘频繁交换数据的情况下，最好不用 fprintf 函数和 fscanf 函数，而用 fread 函数和 fwrite 函数。

14.4.4　其他读写函数

1. getw 函数和 putw 函数

大多数 C 编译系统都提供另外两个函数，即 getw 和 putw，用来对磁盘文件读写一个字。

getw 函数的格式为：

```
getw(fp);
```

其中，fp 是一个 FILE 型指针。它的作用是从指向的文件中读取一个字，如果文件结束或出错，则返回 -1。

例如：

```
i=getw(fp);
```

其作用是从磁盘文件读一个整数到内存，赋给变量 i。

putw 函数的格式为：

```
putw(i,fp);
```

其中，i 是一个整型变量，fp 是一个 FILE 型指针。putw 函数的作用是将 i 的值输出到 fp 指向的文件中去。

例如：

```
putw(10,fp);
```

其作用是将整数 10 输出到 fp 指向的文件。

2. 读写其他类型数据

用 ANSI C 提供的 fread 函数和 fwrite 函数读写任何类型的数据都是十分方便的。如果所用的系统不提供这两个函数，则用户可以自己定义所需函数。

例如，可以定义一个向磁盘文件写一个实数（用二进制方式）的函数 putfloat，即

```
putloat(float num,FILE *fp)
{char *s;
  int count;
```

```
    s=(char *)&num;
    for(count=0;count<4;count++)
    putc(s[count],fp);
}
```

同样也可以编写出读写任何类型数据的函数。

3. fgets 函数和 fputs 函数

fgets 函数的作用是从指定文件读入一个字符串。其格式为：

```
fgets(str,n ,fp);
```

其中，str 是一个字符数组；n 为要求得到的字符个数；fp 是一个 FILE 型指针。函数返回值为 str 首地址。

fputs 函数的作用是向指定的文件输出一个字符串。其格式为：

```
fputs(str,fp);
```

其中，str 是一个字符串常量、字符数组名或字符型指针；fp 是一个 FILE 型指针。

14.5 文件定位

文件中有一个位置指针，指向当前的读写位置。如果顺序读写一个文件，每次读写一个字符，则读写完一个字符后，该位置指针自动移动指向下一个字符。如果想改变这样的规律，强制使位置指针指向其他指定的位置，则可以用有关函数。

1. fseek 函数

fseek 函数用来把文件位置指针移动指定的位置，以便从当前位置读写文件。函数的调用形式为：

```
fseek(fp,offset,origin);
```

其中，fp 为 FILE 型指针；offset 为移动的位移量，以字节为单位，是一个长整型数，ANSI C 标准规定在数字的末尾加一个 L；origin 表示移动位移量的基准点，用 0，1 或 2 代替，0 代表“文件开始”，1 代表“当前位置”，2 代表“文件末尾”。origin 也可以用表示符来表示。SEEK _ SET 表示“文件开始”，SEEK _ CUR 表示“当前位置”，SEEK _ END 表示“文件末尾”。

例如：

```
fseek(fp,100L,0);    /*将位置指针移到离文件头 100 个字节处*/
fseek(fp,50L,1);     /*将位置指针移到离当前位置 50 个字节处*/
fseek(fp,-10L,2);    /*将位置指针从文件末尾处向后退 10 个字符*/
```

注意：fseek 函数一般用于二进制文件，因为文本文件要发生字符转换，所以计算位置时往往会发生混乱。

2. rewind 函数

rewind 函数的作用是使位置指针重新返回文件的开头。其调用形式为：

```
rewind(fp);
```

此函数没有返回值。

3. ftell 函数

ftell 函数的作用是得到流式文件中的当前位置，用相对于文件开头的位移量来表示。ftell 函数的调用形式为：

```
ftell(fp);
```

ftell 返回文件的当前位置相对于文件开始的字节数，如果出错，则返回 -1L。例如：

```
i= ftell(fp);
if(i== -1L)printg("error");
```

变量 *i* 存放在当前位置，如调用函数出错，则输出“error”。

14.6　出错的检测

C 标准提供了一些函数用来检查输入输出函数调用中的错误。

1. ferror 函数

在调用各种输入输出函数（如 putc，getc，fread，fwrite 等）时，如果出现错误，则除了函数返回值有所反映外，还可以用 ferror 函数检查。ferror 函数的一般调用格式为：

```
ferror(fp);
```

当 ferror 返回值为 0（假）时，则表示未出错；如果返回一个非零值，则表示出错。

注意：同一个文件每一次调用输入输出函数，均产生一个新的 ferror 函数值；在执行 fopen 函数时，ferror 函数的初始值自动置为 0。

2. clearerr 函数

该函数的作用是使文件错误标志和文件结束标志置为 0。假设在调用一个输入输出函数时出现错误，则 ferror 函数值为一个非零值。在调用 clearerr（fp）后，ferror（fp）的值变成 0。

只要出现错误标志，就一直保留，直到对同一文件调用 clearerr 函数或 rewind 函数，或任何其他一个输入输出函数。

3. feof 函数

如果处理的是二进制文件而非 ASCII 文件，则读入某一个字节中的二进制数据的值有可能是 -1（而这又恰好是 EOF 的值），但这个 -1 不是文件结束符。这时应该使用由 ANSI C 提供的一个 feof（）函数来判断文件是否真的结束。

feof 函数的形式为：

```
feof(fp);
```

fp 是文件指针。feof（）函数用来检查文件是否结束。如果文件结束，则返回非 0；否则，返回 0。

小结

C 语言的应用几乎都要用文件。有时应用程序需要的输入数据保存在文件中，有时用文

件保存应用程序的输出数据，有时用文件保存应用程序的中间结果。所以在C语言中，对文件的处理是十分重要的。

本章包括文件的基础知识和与文件有关的函数两部分内容。

在文件的基础知识部分中，主要介绍的是文件的基本概念、C语言文件及文件类型指针。在C语言中，文件是一种“数据流”，文件结构为流式文件结构，采用缓冲文件系统，通过文件类型的指针访问文件。

在与文件有关的函数部分中，主要介绍的是文件打开与关闭函数、文件的读写函数、文件定位函数及文件检测函数。需要重点掌握的是文件打开和关闭函数及文件的读写函数。

在C语言中，文件的各种操作是通过函数实现的，而且有的函数（如printf、scanf函数）是通过调用文件函数而得到的。

习题

1. C语言文件的特点是什么？为什么文件输入输出中要采用“缓冲技术”？利用缓冲区输入输出的过程如何？

2. 什么是文件型指针？通过文件指针访问文件有什么好处？

3. 文件使用过程中为什么要打开和关闭？

4. 从键盘输入一个字符串，将其中的小写字母全部转换成大写字母，然后输出到一个磁盘文件“test.txt”中保存。输入的字符串以“!”结束。

5. 有两个磁盘文件“A”和“B”，各存放一行字母，今要求把这两个文件中的信息合并（按字母顺序排列），输出到一个新文件“C”中去。

6. 有5个学生，每个学生有3门课的成绩，从键盘输入学生数据（包括学生号、姓名、三门课成绩），计算出平均成绩，将原有数据和计算出的平均分数存放在磁盘文件“stud”中。

7. 将上题“stud”文件中的学生数据，按平均分进行排序处理，将已排序的学生数据存入一个新文件“stud_sort”中。

8. 综合题。

（1）编程建立一个通讯录文件address，包括姓名、性别、年龄、职务、单位、电话、E-mail地址。

（2）读出通讯录中的信息，按年龄由大到小排序，然后将排序后的信息显示出来，并另存入一个文件address2中。

（3）在address文件中将姓名、电话两项信息抽出来，建立一电话号码表文件tel并显示该文件内容。

（4）根据姓名，从address文件中删除某个人的信息。

附录 A　运算符和结合性

表 A-1　运算符表

优先级	运算符	含义	要求运算对象的个数	结合方向
1	() [] → .	圆括号 下标运算符 指向结构体成员运算符 结构体成员运算符		自左至右
2	! ~ ++ -- - (类型) * & sizeof	逻辑非运算符 按位取反运算符 自增运算符 自减运算符 负号运算符 类型转换运算符 指针运算符 取地址运算符 长度运算符	1 (单目运算符)	自右至左
3	* / %	乘法运算符 除法运算符 求余运算符	2 (双目运算符)	自左至右
4	+ -	加法运算符 减法运算符	2 (双目运算符)	自左至右
5	<< >>	左移运算符 右移运算符	2 (双目运算符)	自左至右
6	< <= > >=	关系运算符	2 (双目运算符)	自左至右
7	== !=	等于运算符 不等于运算符	2 (双目运算符)	自左至右
8	&	按位与运算符	2 (双目运算符)	自左至右
9	^	按位异或运算符	2 (双目运算符)	自左至右
10	\|	按位或运算符	2 (双目运算符)	自左至右
11	&&	逻辑与运算符	2 (双目运算符)	自左至右
12	\|\|	逻辑或运算符	2 (双目运算符)	自左至右
13	? :	条件运算符	3 (三目运算符)	自右至左
14	= += -= *= /= %= >>= <<= &= ^= \|=	赋值运算符	2 (双目运算符)	自右至左
15	,	逗号运算符（顺序求值运算符）		自左至右

说明：

同一优先级的运算符的运算次序由结合方向决定。例如，“*”与“/”具有相同的优先级别，其结合方向为自左至右。因此，3*5/4 的运算次序是先乘后除。“-”和“++”为同优先级，结合方向为自右至左。因此，-i++相当于-(i++)。

不同的运算符要求不同的运算对象个数，如“+（加）”和“-（减）”为双目运算符，要求在运算符两侧各有一个运算对象（如 3+5、8-3 等）。而“++”和“-（负号）”运算符是一元运算符，只能在运算符的一侧出现一个运算对象（如-a、i++、i--、(float) I、sizeof（int）、*p 等）。条件运算符是 C 语言中惟一的一个三目运算符，如 x? a: b。

从表 A-1 中可以大致归纳出各类运算符的优先级：

初等运算符（）[]→。

↓

单目运算符

↓

算术运算符（先乘/除，后加/减）

↓

关系运算符

↓

逻辑运算符（不包括!）

↓

条件运算符

↓

赋值运算符

↓

逗号运算符

以上的优先级别由上到下递减。初等运算符优先级最高，逗号运算符优先级最低。位运算符的优先级比较分散，有的在算术运算符之前（如~），有的在关系运算符之前（如<<和>>），有的在关系运算符之后（如 &、^、|）。为了容易记忆，使用位运算符时可加圆括号。

附录B　C库函数

库函数并不是C语言的一部分。它是由人们根据需要编制并提供用户使用的。每一种C编译系统都提供了一批库函数，不同的编译系统所提供的库函数的数目和函数名及函数功能是不完全相同的。ANSI C标准提出了一批建议提供的标准库函数，包括了目前多数C编译系统所提供的库函数，但也有一些是某些C编译系统未曾实现的。考虑到通用性，本书列出了ANSI C标准建议提供的、常用的部分库函数。对多数C编译系统，可以使用这些函数的绝大部分。由于C库函数的种类和数目很多（如屏幕和图形函数、时间日期函数、与系统有关的函数等，每一类函数又包括各种功能的函数），限于篇幅，本附录不能全部介绍，只从教学需要的角度列出最基本的。读者在编制C程序时可能要用到更多的函数，请查阅所用系统的手册。

1. 数学函数

数学函数见表B-1。使用数学函数时，应该在该源文件中使用以下命令，即

```
# include <math.h>或 #include "math.h"
```

表B-1　数学函数

函数名	函数原型	功能	返回值	说明
abs	int abs (int x)	求整数 x 的绝对值	计算结果	
acos	double acos (double x)	计算 arccos（x）的值	计算结果	x 应在 $-1\sim1$ 范围内
asin	double asin (double x)	计算 arcsin（x）的值	计算结果	x 应在 $-1\sim1$ 范围内
atan	double atan (double x)	计算 arctan（x）的值	计算结果	
atan2	double atan2 (double x, double y)	计算 arctan（x/y）的值	计算结果	
cos	double cos (double x)	计算 cos（x）的值	计算结果	x 的单位为弧度
cosh	double cosh (double x)	计算 x 的双曲余弦 cosh（x）的值	计算结果	
exp	double exp (double x)	计算 e^x 的值	计算结果	
fabs	double fabs (double x)	求 x 的绝对值	计算结果	
floor	double floor (double x)	求出不大于 x 的最大整数	该整数的双精度实数	
fmod	double fmod (double x, double y)	求整除 x/y 的余数	返回余数的双精度数	
frexp	double frexp (double val, int * eptr)	把双精度数 val 分解为数字部分（尾数）x 和以2为底的指数 n，即 val = $x * 2^n$，n 存放在 eptr 指向的变量中	返回数字部分 x $0.5\leqslant x<1$	

续表

函 数 名	函数原型	功 能	返 回 值	说 明
log	double log (double x)	求 $\log_e x$，即 $\ln x$	计算结果	
log10	double log10 (double x)	求 $\log_{10} x$，即 $\lg x$	计算结果	
modf	double modf (double val, double * iptr)	把双精度数 val 分解为整数部分和小数部分并把整数部分存到 iptr 指向的单元	val 的小数部分	
pow	double pow (double x, double y)	计算 x^y 的值	计算结果	
rand	int rand (void)	产生 -90～32 767 间的随机整数	随机整数	
sin	double sin (double x)	计算 sin（x）的值	计算结果	x 单位为弧度
sinh	double sinh (double x)	计算 x 的双曲正弦函数 sinh（x）的值	计算结果	
sqrt	double sqrt (double x)	计算 x 的平方根	计算结果	$x \geqslant 0$
tan	double tan (double x)	计算 tan（x）的值	计算结果	x 单位为弧度
tanh	Double tanh (double x)	计算 x 的双曲正切函数 tanh（x）的值	计算结果	

2. 字符函数和字符串函数

ANSI C 标准要求在使用字符串函数时要包含头文件“string.h”，使用字符函数时要包含头文件“ctype.h”。字符函数和字符串函数见表 B-2。有的 C 编译不遵循 ANSI C 标准的规定，而用其他名称的头文件，使用时应查有关手册。

表 B-2 字符函数和字符串函数

函 数 名	函数原型	功 能	返 回 值	包含文件
isalnum	int isalnum (int ch)	检查 ch 是否为字母（alpha）或数字（numeri）	是，返回 1；不是，返回 0	ctype.h
isalpha	int isalpha (int ch)	检查 ch 是否为字母	是，返回 1；不是，返回 0	ctype.h
iscntrl	int incntrl (int ch)	检查 ch 是否为控制字符（其 ASCII 码在 0 和 0x1f 之间）	是，返回 1；不是，返回 0	ctype.h
isdigit	int isdigit (int ch)	检查 ch 是否为数字	是，返回 1；不是，返回 0	ctype.h
isgraph	int isgraph (int ch)	检查 ch 是否可打印字符（其 ASCII 码在 0x21～0x7e 之间），不包括空格	是，返回 1；不是，返回 0	ctype.h
islower	int islower (int ch)	检查 ch 是否为小写字母	是，返回 1；不是，返回 0	ctype.h
isprint	int ispirnt (int ch)	检查 ch 是否为可打印字符（包括空格），其 ASCII 在 0x21 到 0x7e 之间	是，返回 1；不是，返回 0	ctype.h
ispunct	int ispunct (int ch)	检查 ch 是否为标点字符（不包括空格），即除字母、数字和空格以外的所有可打印字符	是，返回 1；不是，返回 0	ctype.h
isspace	int isspace (int ch)	检查 ch 是否为空格、跳格符（制表符）或换行符	是，返回 1；不是，返回 0	ctype.h

续表

函数名	函数原型	功能	返回值	包含文件
isupper	int isupper (int ch)	检查 ch 是否为大写字母（A～Z）	是，返回1；不是，返回0	ctype.h
isxdigit	int isxdigit (int ch)	检查 ch 是否为一个十六进制数字字符（即0～9，或A～F，或a～f）	是，返回1；不是，返回0	ctype.h
strcat	char * strcat (char * str1, char * str2)	把字符串 str2 接到 str1 后面，str1 最后面的'\0'被取消	str1	string.h
strchr	char * strchr (char * str, int ch)	找出 str 指向的字符串第一次出现字符 ch 的位置	返回指向该位置的指针，如找不到，则返回空指针	string.h
strcmp	int strcmp (char * str1, char * str2)	比较两个字符串 str1、str2	str1 < str2，返回负数 str1 = str2，返回0 str1 > str2，返回正数	string.h
strcpy	char strcpy (char * str1, char * str2)	把 str2 指向的字符串复制到 str1 中去	返回 str1	string.h
strlen	unsigned int strlen (char * str)	统计字符串 str 中字符的个数（不包括终止符'\0'）	返回字符个数	string.h
strstr	char * strstr (char * str1, char * str2)	找出 str2 字符串在 str1 字符串中第一次出现的位置（不包括 str2 的字符串结束符）	返回该位置的指针，如找不到，则返回空指针	string.h
tolower	int tolower (int ch)	ch 字符转换为小写字母	返回 ch 所代表的字符的小写字母	string.h
toupper	int toupper (int ch)	将 ch 字符转换成大写字母	与 ch 相应的大写字母	ctype.h

3. 输入输出函数

凡用以下的输入输出函数，则应该使用#include <stdio.h>把 stdio.h 头文件包含到源程序文件中。输入输出函数见表 B-3。

表 B-3 输入输出函数

函数名	函数原型	功能	返回值	说明
clearerr	void clearerr (FILE * ftp)	清除文件指针错误	无	
close	int close (int ftp)	关闭文件	关闭成功返回0；不成功，返回-1	非 ANSI 标准
creat	Int creat (char * filename, int mode)	以 mode 所指定的方式建立文件	成功返回正数；否则返回0	非 ANSI 标准
eof	int eof (int fd)	检查文件是否结束	如遇文件结束，则返回1	非 ANSI 标准
fclose	int fclose (FILE * ftp)	关闭 fp 所指的文件，释放文件缓冲区	有错返回非0；否则返回0	
feof	int feof (FILE * ftp)	检查文件是否结束	如遇文件结束，则返回非零值；否则，返回0	

续表

函 数 名	函数原型	功 能	返 回 值	说 明
fgetc	int fgetc (FILE * fp)	从 fp 所指定的文件中取得下一个字符	返回所得到的字符，若读入出错，则返回 EOF	
fgets	char * fgets (char * buf, int n, FILE * fp)	从 fp 指向的文件读取一个长度为 ($n-1$) 的字符串，存入起始地址为 buf 的空间	返回地址 buf，若遇文件结束或出错，则返回 NULL	
fopen	FILE * fopen (char * filename, char * mode)	以 mode 指定的方式打开名为 filename 的文件	如成功，则返回一个文件指针（文件信息区的起始地址）；否则，返回 0	
fprint	int fprintf (FILE * fp, char * format, args, ...)	把 args 的值以 format 指定的格式输出到 fp 所指定的文件中	实际输出的字符数	
fputc	int fputc (char * str, FILE * fp)	将字符 ch 输出到 fp 指向的文件中	如成功，则返回该字符；否则，返回非 0	
fputs	int fput (char * str, FILE * fp)	把 str 指向的字符串输出到 fp 所指定的文件	返回 0，若出错，则返回非 0	
fread	int fread (char * pt, unsigned size, unsiged n, FILE * fp)	从 fp 所指定的文件中读取长度为 size 的 n 个数据项，存到 pt 所指向的内存区	返回所读的数据项个数，如遇文件结束或出错，则返回 0	
fscanf	int scanf (FILE * fp, char format, args, ...)	从 fp 指定的文件中按 format 给定的格式将输入数据送到 args 所指向的内存单元（args 是指针）	已输入的数据个数	
fseek	int fseek (FILE * fp, long offset, int base)	将 fp 所指向的文件的位置指针移到以 base 所指出的位置为基准、以 offset 为位移量的位置	返回当前位置；否则，返回 -1	
ftell	long ftell (FILE * tp)	返回 fp 所指向的文件中的读写位置	返回 fp 所指向的文件中的读写位置	
fwrite	int fwrite (char * ptr, usigned size, unsigned n, FILE * fp)	把 ptr 所指向的 n × size 个字节输出到 fp 所指向的文件中	写到 fp 文件中的数据项的个数	
getc	int getc (FILE * fp)	从 fp 所指向的文件中读入一个字符	返回所读的字符，若文件结束或出错，则返回 EOF	
getchar	int getchar (void)	从标准输入设备中读取下一个字符	所读字符。若文件结束或出错，则返回 -1	
getw	int getw (FILE * fp)	从 fp 所指向的文件读取下一个字（整数）	输入的整数。如文件结束或出错，则返回 -1	非 ANSI 标准函数
open	int open (char * filename, int mode)	以 mode 指出的方式打开已存在的名为 filename 的文件	返回文件号（正数），如打开文件失败，则返回 -1	非 ANSI 标准函数

续表

函数名	函数原型	功能	返回值	说明
printf	int printf (char * format, args, ...)	按format指向的格式字符串所规定的格式，将输出表列args的值输出互标准输出设备	输出字符的个数，若出错，则返回负数	format可以是一个字符串，或字符数组的起始地址
putc	int putc (int ch, FILE * fp)	把一个字符ch输出到fp所指的文件中	输出的字符ch，若出错，则返回EOF	
putchar	int putchar (char ch)	把字符ch输出到标准输出设备	输出的字符ch，若出错，则返回EOF	
puts	int puts (char * str)	把str指向的字符串输出到标准输出设备，将'\0'转换为回车换行	返回换行符，若失败，则返回EOF	
putw	int putw (int w, FILE * fp)	将一个整数w（即一个字）写到fp指向的文件中	返回输出的整数，若出错，则返回EOF	非ANSI标准函数
read	int read (int fd, char * buf, unsigned count)	从文件号fd所指示的文件中读count个字节到由buf指示的缓冲区中	返回真正读入的字节个数，如遇文件结束，则返回0；出错，则返回-1	非ANSI标准函数
rename	int rename (char * oldname, char * newname)	把由oldname所指的文件名，改为由newname所指的文件名	成功返回0；出错返回-1	
rewind	void rewind (FILE * fp)	将fp指示的文件中的位置指针置于文件开头位置，并清除文件结束标志和错误标志	无	
scanf	int scanf (char * format, args, ...)	从标准输入设备按format指向的格式字符串所规定的格式，输入数据给args所指向的单元	读入并赋给agrs的数据个数，遇文件结束，返回EOF；出错，则返回0	args为指针
write	int write (int fd, char * buf, unsigned count)	从buf指示的缓冲区输出count个字符到fd所标志的文件中	返回实际输出的字节数，如出错，则返回-1	非ANSI标准函数

4. 动态存储分配函数

ANSI标准建议设4个有关的动态存储分配函数，即calloc ()、malloc ()、free ()、ealloc ()，见表B-4。实际上，许多C编译系统实现时，往往增加了一些其他函数。ANSI标准建议在“stdlib.h”头文件中包含有关的信息，但许多C编译要求用“malloc.h”而不是“stdlib.h”，使用时应查阅有关的手册。

ANSI标准要求动态分配系统返回void指针。void指针具有一般性。它们可以指向任何类型的数据。但目前有的C编译所提供的这类函数返回char指针，无论以上两种情况的哪一种，都需要用强制类型转换的方法把void或char指针转换成所需的类型。

表 B-4 动态存储分配函数

函 数 名	函数和形参类型	功 能	返 回 值
calloc	void * calloc (unsigned n, unsigned size)	分配 n 个数据项的内存连续空间，每个数据项的大小为 size	分配内存单元起始地址，如不成功，则返回 0
free	void free (void * p)	释放 p 所指的内存区	无
malloc	void * malloc (unsigned size)	分配 size 字节的存储区	所分配的内存地址，如内存不够，则返回 0
realloc	void * realloc (viod * p, unsigned size)	将 p 所指出的已分配内存区的大小改为 size，size 可以比原来分配的空间大或小	返回指向该内存区的指针

附录 C　TurboC 常见的出错信息

Argument list syntax error	参数表语法错误
Argument 1 missing name	参数 1 漏了名字（只写了类型）
Array bounds missing	漏掉了数组的界限符“［“或”］”
Array size too large	数组长度太大
Bad character in parameters	参数中含有不合适的字符
Bad filename format in include directive	包含命令中文件名格式不正确
Call of non function	调用未定义的函数
Call to function with no prototype	调用函数时没有声明函数原型
Cannot modify a const object	不允许修改常量对象
Case outside of switch	case 超出了 switch 语句的范围
Case statement missing	漏掉了 case 语句
Case syntax error	case 语法错误
Code has no effect	代码不起作用，即不可能执行到
Compound statement missing }	复合语句漏掉“}”
Conflicting type modifiers	不一致的类型说明符
Constant expression required	需要常量表达式
Constant out of range in comparison	在比较中常量超出范围
Conversion may lose significant digits	转换时丢失有意义的数字
Could not find file ‘xxx’	找不到 xxx 文件
Declaration missing ;	说明语句漏掉“;”
Declaration need type or storage	说明（变量或结构体成员）漏了类型
Declaraton syntax error	说明语法错误
Default outside of switch	default 出现在 switch 语句之外
Division by zero	用 0 作为除数
Do statement must have while	do-while 语句中缺少 while
Enum syntax error	枚举类型语法错误
Error directive : xxx	错误的编译预处理命令
Error writing output file	写输出文件错误
Expression syntax error	表达式语法错误
Extra parameter in call	函数调用时出现多余的参数
File name too long	文件名太长
Function call missing)	函数调用漏掉了右括号“)”
Function definition out of place	函数定义位置错误（如在函数内定义）

Function should return a value	(非空类型) 函数必须返回一个值
Goto statement missing label	goto 语句漏了标号
Hexadecimal or octal constant too large	十六进制或八进制常量太大
Illegal charater 'x'	非法字符 "x"
Illegal initialization	非法初始化
Illegal octal digit	非法的八进制数字
Illegal pointer subtraction	非法的指针相减
Illegal structure operation	非法的结构体操作：结构体成员引用错
Illegal use of floating point	非法的浮点运算 (如对实数求余等)
Illegal use of pointer	非法使用指针
Incompatible type conversion	不相容的类型转换
Incorrect number format	错误数据格式
Incorrect use of default	default 使用不正确
Invalid indirection	无效的间接运算
Invalid pointer addition	指针相加无效
Lvalue required	需要变量 (如赋值号左边不是变量)
Macro argument syntax error	宏参数语法错误
Mismatched number of parameters in definition	定义中参数个数不匹配
Misplaced break	break 语句放错位置
Misplaced continune	continue 语句放错位置
Misplace else	else 子句放错位置
Must take address of memory location	必须是内存单元的地址
No declaration for function 'xxx'	没有函数 xxx 的说明
Non - portable pointer assigment	不可移的指针 (地址常量) 赋值
Non - portable pointer conversion	不可移动的指针 (地址常量) 转换
Not a valid expression format type	不合法的表达式格式
Not an allowed type	不允许使用的类型
Numeric constant too large	数值常数太大
Out of memory	内存不够用
Parameter 'xxx' is neve used	参数 xxx 从未用到
Pointer required on left side of - >	"- >" 运算符的左边必须是指针
Possible use of 'xxx' before definition	xxx 没有赋值就使用
Possibly incorrect assignment	赋值可能不正确
Redeclaration of 'xxx'	符号 "xxx" 重复定义
Redefinition of 'xxx' is not indentical	xxx 的重复定义不一致
Size of structure or array not known	结构体或数组的大小不确定
Statement missing;	语句漏了分号；
Structure or union syntax error	结构体或共用体语法错误
Subscripting missing]	下标漏了右方括号 "]"

Suspicious pointer conversion	可疑的指针转换
Too few parameters in call 'xxx'	函数调用时实参少于形参
Too many default cases	switch 语句中 default 太多
Too many types in declaration	说明中类型太多
Type mismatch in parameter 'xxx'	参数 xxx 类型不匹配
Type mismatch in redeclaration of 'xxx'	xxx 重定义时类型不匹配
Unable to creat output file 'xxx'	无法建立输出文件 xxx
Unable to open include 'xxx'	无法打开包含文件 xxx
Unable to open input file 'xxx'	无法打开输入文件 xxx
Undefined structure 'xxx'	结构体 xxx 没有定义
Undefined symbol 'xxx'	符号 xxx 没有定义
Unreachable code	无法执行到的代码（如 if 或 while 语句中条件永不成立，语句 1 就不可能被执行）
Unterminated string or character constant	字符串或字符常量漏了半边引号
Void functions may not return value	void 类型的函数不应有返回值
Wrong number of arguments	调用函数时参数个数错
'xxx' not part of structure	xxx 不是结构体的一部分
'xxx' statement missing (	xxx 语句漏了 "("
'xxx' statement missing;	xxx 语句漏了分号 ";"
'xxx' declared but never used	说明了 xxx，但从没有使用
'xxx' is assigned a value which is never used	给 xxx 赋了值，但从未使用过

参 考 文 献

1. 王玉龙. 计算机导论. 北京：电子工业出版社，1997
2. 赵晓东，朱建华. 计算机导论. 北京：电子工业出版社，2001
3. 朱战立，李高和，杨谨全. 计算机导论. 西安：西安电子科技大学出版社，2003
4. 耿国化，索琦. 计算机基础与C语言程序设计. 北京：电子工业出版社，2002
5. 田原等. 计算机导论. 北京：中国水利水电出版社，2001
6. 张雷，郑岩. 计算机导论与程序设计. 北京：北京邮电大学出版社，2002
7. Baber R，MeyerM. 计算机导论. 汪嘉旻译. 北京：清华大学出版社，2000
8. 原忠虎等. 计算机文化基础. 沈阳：东北大学出版社，2000
9. 谭浩强，张玲. 计算机基础知识与基本操作. 北京：清华大学出版社，1999
10. 李秀等. 计算机文件基础 . 第3版. 北京：清华大学出版社，2000
11. 徐惠民，顾淑清. 计算机文化基础（Windows 2000 版）. 北京：人民邮电出版社，2001
12. 谭浩强. C程序设计 . 第2版. 北京：清华大学出版社，1999
13. 徐建民等. C语言程序设计. 北京：电子工业出版社，2002
14. 徐士良. C语言程序设计教程. 北京：人民邮电出版社，2001
15. 李凤云，刘凤华. C/C++程序设计基础. 北京：人民邮电出版社，2003
16. 刘瑞新，丁爱萍，李树东. 计算机组装与维护教程. 北京：机械工业出版社，2001